CONSIGNES SÉRIE 2 : LA GESTION DU PANIER

Le panier d'achats est un élément essentiel de votre site. La méthode proposée se réalise en deux classes et un script, le tout plutôt facile à gérer. Suivez les étapes avec rigueur pour obtenir des résultats fiables.

Vous devriez avoir terminé **le catalogue** avant de vous attaquer aux consignes 2. Si vous avez fait ce qui était demandé, sur un clic du bouton d'ajout au panier dans le catalogue, vous appelez une page nommée **panierGestion.php** en lui transmettant deux paramètres : un qui dit **quoi faire** (**ajout**) et un autre qui donne le numéro du produit.

```
echo "<a href='panier.php?quoiFaire=ajout&noProduit="…
```

Partant de cela, vous devrez implanter les fonctionnalités suivantes :

- ajouter un item au panier;
- modifier la quantité d'un ou de plusieurs items au panier;
- supprimer un item du panier;
- vider le panier;
- afficher le contenu du panier.

Un panier doit garder son contenu au fil de la navigation. Pour réaliser cette persistance vous utiliserez des variables de **session**. Dans la mise au point du script on doit souvent les supprimer : il suffit de fermer **toutes les instances** du fureteur et de le redémarrer pour terminer la « session » puis en débuter une nouvelle, avec des variables « toutes fraîches ».

Pour gagner du temps (paresse à long terme toujours souhaitable), on peut vider toutes les variables de session en programmant un petit script **viderSession.php** qui contient le code suivant :

```
<?php
  session_start();
  $_SESSION = array();
  header("location:index.php");
  exit();
?>
```

Dans votre catalogue, mettez un hyperlien qui pointe vers lui. Ce script vous apportera confort et efficacité. Vous enlèverez le lien avant la remise finale du TP.

ÉTAPE 1 : LE SCRIPT PANIER.PHP

C'est **panierGestion.php** qui a accès aux variables transmises dans le **get**. Essentiellement, il doit

1. obtenir l'accès aux variables de session;
2. instancier un objet BDService;
3. instancier un objet Panier (voir l'étape 3);
4. gérer le paramètre **quoiFaire** du **get** ainsi que le numéro de produit transmis dans l'autre paramètre;
5. afficher le contenu du panier.

Voyons ceci en plus de détails.

1. Pour avoir accès aux variables de session, vous devez appeler l'énoncé **session_start();** Cet énoncé a ceci de particulier qu'il doit être exécuté par PHP **avant** le moindre HTML.

   On a vu plus haut que le contenu du panier sera emmagasiné dans une variable de session. Or celle-ci contiendra des objets de type **Achat** qui sont eux-mêmes dérivés d'objets de type **Produit**. Pour accéder à vos classes dans vos scripts PHP, vous pouvez utiliser un require par classe par ex :

```
<?php
require_once "Classes/produit.php";   //D'abord la classe mère
require_once "Classes/achat.php";     //puis la classe dérivée
session_start();
```

   ou encore la méthode spl_autoload_register() telle que vue en classe. Cette dernière méthode est plus élégante et pratique.

   Ici, on doit spécifier un détail important : il est dit plus haut que le **session_start()** doit être envoyé par le serveur avant tout HTML. Si une de vos classes contient une ligne blanche ou même un seul espace à l'extérieur des balises **<?php** et **?>**, cela devient du HTML transmis et bloque le fonctionnement de session_start. Vous recevez alors un message du style **Warning, header already sent…**.

2. Le script ouvre ensuite la BD en instanciant un objet BDService:
   $maBD = new BDService();.

3. Puis il se crée un panier. Les classes **Achat** et **Panier** sont expliquées plus bas.

```
$panier = new Panier();
```

4. Il gère le **get** par une structure organisée comme ceci **:**

```
if (isset($_GET['quoiFaire'])) {
   switch ($_GET['quoiFaire']) {
      case "ajout": $panier->ajouter($_GET['noProduit']);
                    break;
      case "modification": ...
                    break;
```

```
      case "suppression":  ...
                    break;
      case "vider": ...
                  break;
      default : break;
   } // Fin du switch
} // Fin du if
...
```

5. Enfin, à la suite du **switch**, et après avoir posé le HTML approprié — appel de votre fonction d'en-tête et de celle de la colonne de gauche — ainsi que les spécificités de la page du panier, on aura

```
$panier->afficher();
```

Ultimement, votre page ressemblera à celle-ci (en plus joli).

Ce début de programme pourra être testé dès que vous aurez créé la version originale des deux classes qui suivent. Vous le compléterez ensuite.

ÉTAPE 2 : LA CLASSE ACHAT

Le prochain outil dont vous aurez besoin est une classe pour traiter **un produit** ajouté au panier. Nous en profiterons pour exploiter l'**héritage,** concept clé de l'orienté objet.

En effet, qu'est-ce qu'un achat dans ce site ? C'est un objet ayant toutes les caractéristiques d'un **produit** auquel on ajoute une **quantité commandée** pour éventuellement compléter une transaction.

Vous allez créer une nouvelle classe **Achat** en la dérivant de la classe **Produit**. Elle va ainsi hériter de toutes les variables membres et de toutes les méthodes de **Produit**. Pour les besoins du site, vous allez lui ajouter une variable membre.

Syntaxiquement, cela s'écrit

```
class Achat extends Produit {
   protected $quantite;
```

Dans le concept d'héritage, on dit que toutes les méthodes de la classe mère sont disponibles à la classe fille **à moins que celle-ci ne les redéfinisse**. Si on écrit un constructeur dans la classe **Achat**, il aura donc préséance sur celui de la classe **Produit.** Par contre, celui de la classe mère fait une grande partie du travail. Voici comment profiter de ce constructeur. Si, dans la classe **Produit**, vous avez opté pour un constructeur qui reçoit un tableau, le travail sera particulièrement simple. Sinon, soit vous adaptez ce qui suit, soit vous retournez changer votre classe **Produit** et ses quelques instanciations.

L'objet a donc toutes les variables membres du **Produit** plus une quantité. On pourrait, lors d'un ajout de produit au panier, l'instancier ainsi :

```
$achat = new Achat($ligne); //$ligne est un tableau obtenu de la BD avec les valeurs pour un produit
                            //
```

Cela nous amène au constructeur suivant :

```
public function __construct ($tableau) {
   $this->quantite = 1;     //On démarre l'achat avec une quantité de 1
   parent::__construct($tableau);
}
```

D'une part, on initialise la variable **$quantite** à 1. On pourra la modifier plus tard. D'autre part, on utilise le constructeur de la classe mère (syntaxiquement, cela s'écrit **parent::__construct()**) sans avoir à reproduire du code qu'on a déjà écrit.

Examinez le site de démonstration et créez une méthode **afficher()** pour afficher un achat dans la page du panier.

Vous aurez probablement à ajouter à votre classe deux ou trois autres méthodes de **get()** et de **set()**, selon les besoins.

Sauvegardez, sous le nom **achat.php** dans le répertoire **Classes**. Vous pourrez tester sous peu.

ÉTAPE 3 : LA CLASSE PANIER

La classe **Panier** composera des achats en un tableau qui permettra de gérer de façon cohérente un ensemble d'achats.

Ce sera une des classes les plus complexes du site. En y allant une étape à la fois, et en suivant soigneusement les consignes, votre panier sera robuste et fiable.

Comme pour le catalogue, cette classe possède une seule variable membre. Plutôt que de compter sur des valeurs passées en paramètres pour initialiser les objets, elle fera référence aux variables **superglobales** pour assurer la persistance du panier d'une page à une autre.

LE CONSTRUCTEUR

Démarrez avec ce code :

```
class Panier {
  private $tabAchats;
  public function __construct() {
     $this->tabAchats =array();               //On commence par un panier vidé de ses achats
     if (isset($_SESSION['panier'][0]))      //S'il y a au moins un premier item déjà dans le panier conservé
        $this->tabAchats=$_SESSION['panier']; //on y met le contenu déjà mémorisé
  }
```

Si cela peut sembler mystérieux pour le moment, voici quelques explications avec promesse d'éclaircissements plus tard.

À chaque passage dans la page du panier, votre script va enregistrer les données qu'il contient dans une variable de session sous le nom $_SESSION['panier'], cette variable désigne un tableau d'achats. Les achats seront dans $_SESSION[['panier'][0], $_SESSION[['panier'][1], etc.

Le constructeur démarre avec un panier vide et si la variable superglobale $_SESSION['panier'][0] a un contenu, c'est-à-dire qu'il y a au moins un item dans le panier, on copie l'ensemble de ce panier dans la variable membre **$panier**. Encore une fois, on constate que la syntaxe de PHP est très simple: pour copier un tableau, il suffit de faire une seule affectation.

L'AFFICHAGE DU PANIER, VERSION INITIALE

Créez une première version d'une méthode **afficher();**

Commencez par déterminer si le panier est vide ou non : `$nbAchats=count($this-> tabAchats);`

Vous traiterez le cas du panier vide puis vous écrirez la boucle qui affiche tous les achats ainsi :

```
if ($nbAchats == 0)
    echo "Votre panier est vide ";
```

```
else {
   for($i=0; $i < $nbAchats; $i++) {
        $article=$this->tabAchats [$i];
        $article->afficher($i);  //On verra plus loin pourquoi le $i
...
```

Par la suite, vous reviendrez sur cette méthode pour qu'elle gère les totaux.

L'AJOUT AU PANIER

Vous devez maintenant créer une méthode **ajouter()** dans la classe panier. Que contient le panier? des achats. Et que sont les achats ? des produits avec une quantité commandée. Comment sait-on quel est le produit demandé par le client ? il a été codé dans le **get** quand on a cliqué sur le bouton d'ajout au panier du catalogue et il a été inclus plus haut dans l'énoncé `$panier->ajouter($_GET['noProduit']);` Dans une première version, voici ce que la méthode doit accomplir :

À partir du numéro de produit, on doit

- aller chercher les informations complètes de ce produit dans la BD;
- avec ces informations, on instancie un objet **Achat** qu'on ajoute au panier;.
- on « écrit » le panier dans le tableau de session pour assurer sa persistance.

```
...
$achat = new Achat (…);
$this->tabAchats = $achat;
$_SESSION['panier'] = $this->tabAchats;
```

Si un client demande à nouveau un produit qui est déjà dans son panier, il est important de ne pas l'ajouter au panier mais plutôt d'augmenter sa quantité de 1.

Avant de faire un ajout, si le panier n'est pas vide, vous devrez d'abord balayer le tableau pour vérifier si le produit y est déjà. Si c'est le cas, vous augmentez sa quantité plutôt que de l'ajouter.

Pour ce faire, vous aurez besoin de lire le ***noProduit*** de vos objets **Achat**. Voici donc l'occasion d'ajouter une méthode **getNoProduit()** à votre classe **Achat**. Semblablement, vous allez avoir besoin d'augmenter la quantité d'un achat déjà présent au panier : deux autres méthodes s'imposent : **getQuantite()** et **setQuantite()**.

Voici à peu près le code tel qu'il apparaîtra dans votre script.

Dans la classe **Achat**,

```
public function getNoProduit() {
    return $this->noProduit;
}
```

```
public function setQuantite($nb) {
    $this->quantite = $nb;
}
...
```

Dans la classe **Panier**, au début de la méthode **ajouter()**

```
$dejaPresent = false;
$nbAchats = count($this->tabAchats); //nb d'articles dans le panier
if ($nbAchats > 0 ) {
    for($i = 0; $i < $nbAchats; $i++)
        if ($noProduit == $this->tabAchats[$i]->getNoProduit()) {  //$noProduit est le numéro à ajouter
            $dejaPresent = true;                                   //passé en paramètre à la méthode
            break;
        }
}
if ($dejaPresent) {
    $quantite = $this->tabAchats[$i]->getQuantite();  //Lecture de la quantite actuelle
    $this->tabAchats[$i]->setQuantite($quantite + 1);
else
... suite du traitement (lire le produit et l'ajouter au panier)
```

L'AFFICHAGE DU PANIER, LA SUITE

Une fois que votre ajout au panier fonctionne, vous allez ajouter à la méthode **afficher()** les éléments suivants :

- un hyperlien de suppression,
- un champ pour changer la quantité,
- un bouton de modification comme dans le site de démonstration.

Branchez vous sur le site de démonstration, ajoutez quelques items au panier. Passez le pointeur de souris sur les liens de suppression. Vous devriez constater qu'il y a un paramètre numérique qui identifie la ligne à supprimer.

Affichez le code source. Constatez que les champs de saisie portent le nom **quantite0**, **quantite1**, etc. Vous avez déjà, lors d'un exercice précédent, généré des noms variables de champs (`name='quantite'.$i.)`, cette technique revient ici dans toute son utilité. Elle vous permettra de générer autant de champs de saisie qu'il y a de produits dans le panier, chacun avec un nom différent des autres.

Vous devriez constater aussi que les champs de saisie et le bouton de modification **sont dans un formulaire** puisqu'on veut transmettre des valeurs. Comme vous l'avez fait pour le catalogue, ce formulaire s'appelle lui-même.

Cependant, nous avons ici un cas particulier dans lequel nous utilisons le paramètre **get** pour déterminer **quoi faire** mais nous aurons aussi besoin de transmettre les quantités à modifier. Or, on ne peut pas « cumuler » des valeurs **get** du formulaire avec celle du paramètre **action**. On le fera en transmettant simultanément un contenu dans le **post**. Codez donc votre balise **<form>** de la façon suivante :

```
<form method='post' action='panier.php?quoiFaire=modification'>
```

Plus haut, les consignes précisent que l'affichage d'une ligne s'appelle par `$article->afficher($i);` C'est ici que la valeur **$i** passée en paramètre servira à compléter le lien de suppression ainsi que le nom du champ de saisie pour la modification

Si on suppose que le paramètre de réception est **$num**, l'hyperlien de suppression doit pointer vers la **page elle-même** de la manière suivante :

```
<a href='panier.php?quoiFaire=suppression&no=$num '>Supprimer ce produit </a>;
```

Pour chaque élément également, vous incluez une balise de champ de saisie qui permettra de modifier les quantités. Vous la codez ainsi :

```
echo "<input type='text' name='quantite".$num."' size='3' value='".$this->quantite."'/>";
```

Notez enfin que la fonction **number_format()** vous sera encore utile pour l'affichage du prix avec deux décimales. Testez avec des produits que vous achetez en quantité de 10.

Attention, il ne faut utiliser cette fonction que lors de l'affichage, ne l'utilisez pas lors de l'initialisation de variables tel que

```
$this->prix=number_format($this->prix,2);
```

en effet **number_format()** introduit, par défaut, des espaces pour les milliers.

Enfin, mettez dans le formulaire un bouton **Submit**.

Testez. N'oubliez pas que les variables de **session** sont « persistantes ». Quand vous avez un problème, il faudra les ré-initialiser. Assurez-vous que l'ajout fonctionne et que la liste des produits est correcte.

Avant d'aller plus loin, cliquez sur un hyperlien de suppression. La page devrait s'appeler elle-même et vous devriez voir, dans la ligne d'adresse, **…?quoiFaire=suppression&no=*x***

Cliquez aussi sur le bouton de modification et vous devriez voir **…?quoiFaire=modification**

Il ne reste plus qu'à coder les deux autres fonctionnalités.

Les taxes

Au Québec, deux taxes s'appliquent au prix de base. La TPS et la TVQ.

Les taux de taxation ne varient pas fréquemment ce qui justifie de les déclarer comme constantes dans un script inclus par tous les autres scripts. Par exemple, le fichier contenant votre fonction de l'entête:

```
define("TPS",0.05);
define("TVQ",0.09975);
```

```
function enTete() {...}
```

Vous pourrez utiliser ces constantes partout et, en cas de changement d'une taxe, vous n'aurez qu'une seule ligne de code à modifier pour l'ensemble de votre site.

LA MODIFICATION DES QUANTITÉS

Passons au **case "modification"** de votre **switch()**. On veut donc coder la méthode **modifier()**.

Essentiellement, il suffit ici de boucler sur les champs passés par **$_POST** (rappelez-vous que vous avez codé cette méthode dans la balise <form> parce que le **get** était déjà occupé par **?quoiFaire=modification**).

Le nom des champs a été construit dynamiquement (**quantite0**, **quantite1**, etc.)

```
$nbAchats = count($this->tabAchats);
for($i=0; $i < $nbAchats; $i++) {
     valider la quantité du champ $_POST['quantite'.$i] et si elle est bonne
        $this->tabAchats[$i]->setQuantite(...);
     sinon préparer un message d'erreur
//Conserver le panier
$_SESSION['panier'] = $this->tabAchats;
```

C'est tout. Si vous avez codé comme demandé, le panier devrait s'afficher avec les nouvelles quantités.

Avant de passer à la dernière étape, ajoutez une validation pour bloquer les valeurs négatives ou non numériques. Une expression régulière est idéale — chiffres seulement — et vous n'acceptez que les valeurs supérieures à zéro.

Pour pouvoir afficher un message d'erreur comme sur le site de démonstration, la méthode **modifier()** doit retourner une valeur qui indique si oui ou non tout était correct. Ensuite, en vous basant sur ce résultat et en regardant comment le site de démonstration traite ce cas, vous devriez trouver comment le coder.

LA SUPPRESSION D'UN ÉLÉMENT DU PANIER

Un peu plus complexe que la modification, le code de cette partie doit être testé soigneusement.

L'hyperlien codé à la première partie envoie l'indice dans le panier du produit à supprimer. Pour effectuer cette opération,

- vous allez écraser ce produit avec ceux qui le suivent dans le panier;
- vous allez désallouer le dernier élément du tableau éliminant de ce fait le produit et rendant le dernier élément disponible pour un ajout.

Supposons que la méthode reçoit en paramètre le numéro de l'élément sous le nom **$num**, correspondant à l'élément que vous voulez écraser, et par une boucle, vous copiez vers le début du tableau. Par exemple

```
//Pour supprimer cet élément du tableau, on l'écrase avec ceux qui suivent
$nbAchats = count($this->tabAchats);
for($i=$num+1; $i < $nbAchats; $i++) {
    $this->tabAchats[$i-1] = $this->tabAchats[$i];
```

Les bornes de votre boucle sont importantes pour fonctionner dans tous les cas. Enfin, pour désallouer le dernier élément, vous écrivez :

```
unset ($this->tabAchats[$nbAchats-1]);
```

Et vous conservez le panier par le même énoncé qu'ailleurs

```
$_SESSION['panier'] = $this->tabAchats;
```

Pour tester, mettez plusieurs items dans votre panier. Supprimez le premier, le dernier, un autre dans le milieu du panier et assurez-vous toujours que le panier reste correctement géré.

LA SUPPRESSION DE TOUT LE CONTENU DU PANIER

Un seul énoncé fera ici le travail… à vous de le découvrir.

Il vous reste, à gérer les prix avec le total de la facture. Un peu de codage et ça y est.

RAFFINEMENT DE LA GESTION DU GET

Sur le site de démonstration, le paramètre **get** n'apparaît pas quand on effectue des opérations sur le panier. C'est pour éviter qu'un utilisateur, s'il tape la touche **F5** ou qu'il clique sur le bouton **Actualiser** ne voie les quantités augmenter. Voici la recette :

Dans le script **panier.php**, avant le **break;** complétez chaque élément du **switch** de la manière suivante :

```
...
header('location:panier.php');
exit();
```

La page se rappelle elle-même. Comme il n'y a pas de valeur transmise pour le paramètre **quoiFaire**, le script passe immédiatement à l'affichage sans rien faire d'autre… et le **get** est vide et on peut actualiser la page autant qu'on le veut, il ne se passe rien d'indésirable.

Avant de tout remettre, assurez-vous que les opérations suivantes fonctionnent correctement :

- Ajoutez un produit plus d'une fois. La quantité commandée devrait augmenter ;

- Ajoutez d'autres produits. Ajoutez un produit déjà dans le panier mais en position 2, en position 3, etc. pour vous assurer que l'ajout se fait bien au bon endroit ;
- Supprimez le premier produit du panier, supprimez le dernier, supprimez dans le milieu du panier. Assurez-vous chaque fois que c'est le bon produit qui disparaît ;
- Mettez des quantités incorrectes comme **rr**, **1a**, **a1**, **7e2** (qui est vu comme numérique par PHP) **0**, **-1**, **a-1**, etc. Aucune ne devrait passer la validation ;
- Mettez une quantité de 10 ou de 100 pour un article et assurez-vous que tous les montants apparaissent avec deux chiffres après le point.