université PARIS DIDEROT PARIS 7

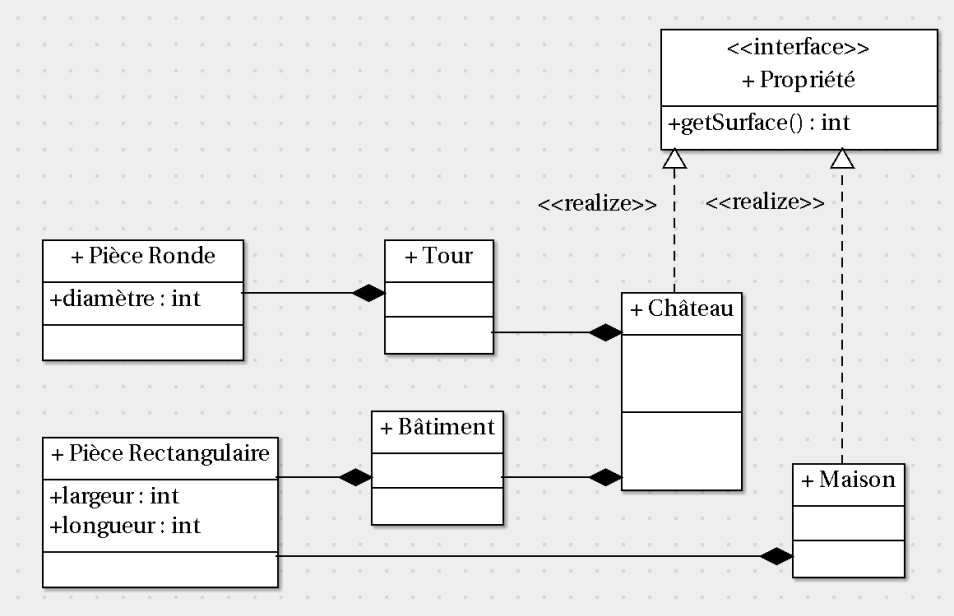
EXAMEN | LOAV — C++ | 18/06/2018 8^{30}–10^{30}

Aucun document ou support autre que le sujet ou les copies d'examen n'est autorisé.
(la copie ou les brouillons du voisin ne sont pas des supports autorisés).
Éteignez impérativement vos mobiles.

1 Exercice

Soit le diagramme UML suivant qui représente la modélisation (une toute petite partie) d'un domaine féodal. Le diagramme est évidemment incomplet. On souhaite pouvoir créer un château à l'aide du programme suivant :

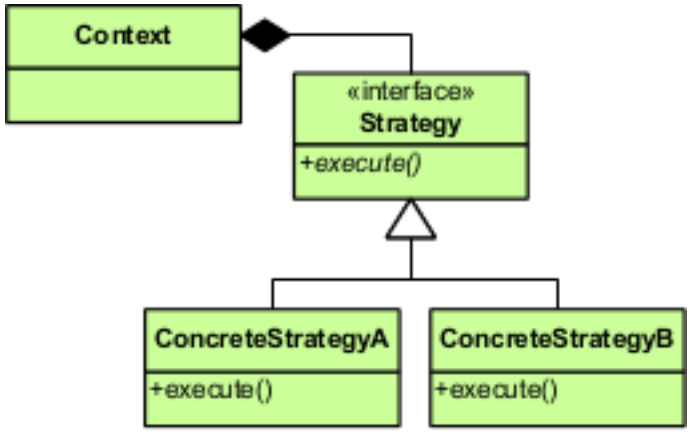
```
int void main() {
  Chateau seigneurie("Château du duc de Paris 7");
  // ajout d'une tour de 10m de diamètre
  seigneurie.addTour(10);
  // ajout de 5 pieces rondes de diametre celui de la tour)
  for (int i=0; i<5; i++) seigneurie.getTour(0).addPiece();
  // ajout d'un batiment de 100m x 20m
  seigneurie.addBatiment(100,20);
  // ajout d'une pièce de 30mx40m
  seigneurie.getBatiment(0).addPiece(30,40);

  cout << seigneurie.getSurface() << endl; // affichage de la surface totale
  cout << seigneurie << endl; // affichage de tout le contenu de la propriété
  cout << seigneurie.getTour(0) << endl; // affichage de tout le contenu de la première tour

  Maison fermette(10,5);
  fermette.addPiece(5,5);
}
```



Créer les différentes classes avec attributs et méthodes permettant d'obtenir les résultats attendus...

2 Exercice

Soit le diagramme UML représentant le pattern strategy :



Utiliser ce pattern pour obtenir l'effet suivant :

> Une personne est dotée d'une humeur qui peut-être Agréable, Ronchon ou Neutre. Lorsqu'une personne est d'humeur Agréable et qu'on la salue elle répond par «Bien le bonjour à vous», Ronchon par «Grmffff» et Neutre par «Bonjour».

— Implémenter ce pattern en C++ de sorte que l'on puisse modifier dynamiquement l'humeur d'une personne.

— Créer une nouvelle humeur Amical qui permet d'obtenir la réponse «Salut !». Attention aucune modification de la classe Personne ne doit intervenir...

3 Exercice

Soit le code du tri QuickSort suivant :

```
void quickSort(int arr[], int left, int right) {
      int i = left, j = right; int tmp; int pivot = arr[(left + right) / 2];
      /* partition */
      while (i <= j) {
          while (arr[i] < pivot) i++;
          while (arr[j] > pivot) j--;
          if (i <= j) {
                tmp = arr[i]; arr[i] = arr[j]; arr[j] = tmp;
                i++;
                j--;
          }
      }
      /* recursion */
      if (left < j) quickSort(arr, left, j);
      if (i < right) quickSort(arr, i, right);
}
```

— créer une version templatée de ce tri pour fonctionner sur n'importe quel type d'élement (tri d'un tableau de chaîne de caractères ou n'importe quoi d'autre).

— créer un type MyInt qui pourra être utilisé dans la fonction templatée

— modifier le type MyInt de sorte que l'on puisse mesurer la complexité de l'algorithme précédent, en comptant le nombre de comparaisons, le nombre d'échanges. Pour aider au comptage du nombre d'échanges on pourra doter le type `MyInt` d'une méthode `MyInt.swap(MyInt &)` permettant d'échanger le contenu de deux `MyInt`. Écrire cette méthode.

4 Exercice

Écrire des classes permettant de construire une salade de fruits de la façon suivante :

```
Fraise f;
Fruit &unFruit = Orange();
Banane b;
Salade s = f+b+unFruit;
cout << s << endl;
```

qui affichera : «Hummm, la bonne salade de fruit avec fraise, banane et orange.»
Écrire les différentes classes (et leurs méthodes, champs, etc) nécessaires.