Question 3 Programmation imperative et concurrente en Go

Le programme `houses.go` permet de simuler une vente aux enchères de propriétés à vendre. Exécuter ce programme afin de voir une simulation avec 7 acheteurs proposant des prix pour l'achat de 3 propriétés mises en vente par 3 vendeurs. Cette simulation est aléatoire donc différents résultats sont obtenus à chaque exécution.

Ce programme définit 5 types principaux:

- `Seller` représentant les vendeurs de propriétés (un vendeur par propriété)
- `Buyer` représentant les acheteurs qui feront des mises pour acheter des propriétés
- `Condo`, `House`, et `TownHouse` représentant les propriétés à vendre

Note: il n'est pas nécessaire d'analyser et de comprendre le code définissant les types `Seller` et `Buyer` afin de répondre aux questions ci-dessous.

La function `main` de ce programme effectue la simulation de la vente aux enchères (*the bidding process*). Ce processus est subdivisé en étapes explicitées ci-dessous.

Répondre aux questions a) à h).
Les numéros de lignes spécifiés font référence au programme `houses.go`

STEP 1: La liste des propriétés à vendre. (ligne 179)

La variable `listings` contient les propriétés à vendre. Dans le programme, `listings` est un slice de pointeurs à des instances de `Condo`.

```
listings := []*Condo{{"Goulburn Ave 1120", 750000, false, 900},
                     {"Summerset Street 10", 950000, false, 850},
                       {"Wilbord Avenue 999", 1250000, false, 1250}}
```

Dans le cas général, `listings` devrait aussi contenir des instances de `House` et `TownHouse`.

a) Types embarqués (Embedding): Créer la structure `ListingInfo` afin de contenir tous les attributs communs à `Condo`, `House` et `TownHouse`, enlever ces attributs dans `Condo`, `House` et `TownHouse` et embarquer à la place la structure `ListingInfo` dans `Condo`, `House` et `TownHouse`.

```
// ListingInfo structure
type ListingInfo struct {
   // code to be added here
   // see line 14 in Go code provided
}
```

b) **Interfaces:** Changer la déclaration de la variable `listings` afin qu'elle soit plutôt un slice de `RealEstate` avec `RealEstate` étant une interface que doivent réaliser les types `Condo`, `House` et `TownHouse` :

```
listings := []RealEstate{
              {"Goulburn Ave 1120", 750000, false, 900},
               {"Summerset Street 10", 950000, false, 850},
                {"Wilbord Avenue 999", 1250000, false, 1250}}
```

c) Ajouter un `House` et un `TownHouse` à `listings` (il y aura donc maintenant 5 propriétés dans la liste)
- Le `TownHouse` a 2 niveaux et est situé à "Elgin 123" pour un prix de $2100000
- Le `House` est à "Maplewood 889" à un prix de $850000 avec un lot de 50 x 110

STEP 2: Les vendeurs de propriétés (line 185)

```
sellers := []*Seller{NewSeller("Eve", listings[0]),
                     NewSeller("Monica", listings[1]),
                     NewSeller("Ramon", listings[2])}
```

d) Ajouter un vendeur pour le `House` et un vendeur pour le `TownHouse` de la question c) (il y aura donc maintenant 5 vendeurs).
- Paul le vendeur du `TownHouse`
- Mary le vendeur du `House`

Note: si vous n'avez pas pu répondre à la question c), donner simplement la réponse à d) en placant le code qui devrait être ajouté en commentaires.

STEP 3: Les acheteurs (line 190)

Il n'y a rien à faire ici.

```
buyers := []*Buyer{NewBuyer("Zara"), NewBuyer("Jim"),
                   NewBuyer("Claude"), NewBuyer("Emilie"),
                   NewBuyer("Amelie"), NewBuyer("Ali")}
```

STEP 4: Les enchères (Bidding process) (line 194)

Le processus d'enchères est actuellement complétement séquentiel, il doit être changé afin d'être concurrent.

Note: Vous pouvez répondre à cette question avec ou sans la structure `ListingInfo` ou l'interface `RealEstate` introduites dans les questions précédentes.

e) Go Routines:

Dans la boucle `for` du processus d'enchères (line 193), chaque acheteur (*buyer*) essai d'acheter les propriétés à vendre; ceci se fait un acheteur après l'autre (séquentiellement). Introduire une routine go (possiblement sous forme d'une fonction lambda) de façon à ce que chaque acheteur procède aux enchères (*bid*) concurremment.

```
for _, buy := range buyers { // for each buyer
    // Buyers need to bid concurrently
   // introduce a Go rountine here
```

f) **Channels:**
La seconde boucle `for` (line 200) permet à un acheteur de proposer des prix croissants pour chacunes des propriétés:

```
// loop by generating different amount
for amount, valid := buy.nextBid(); valid;
                          amount, valid = buy.nextBid() {
```

Cette boucle s'arrête si un vendeur accepte le montant proposé par l'acheteur ou si l'acheteur ne veut plus proposer un montant plus élevé.

Note: pour répondre à la question vous n'avez pas à comprendre comment les prix successivement proposés (en utilisant `nextBid`) sont générés.

Dans la version iterative courante, un acheteur vérifie si un vendeur accepte son offre en appelant la fonction `acceptBid` (line 207):

```
// Is bid accepted? - if true is returned then close deal
if s.acceptBid(amount) {
```

Mais dans le type `Seller` (ligne 116) on retrouve le channel `OfferChan` destine à recevoir les montants proposés et le channel booléen `ResponseChan` à travers lequel la réponse du vendeur est envoyé. Chaque fois qu'un vendeur est créé, la function go suivante est aussi lancée (line 142):

```
go func() {
     for {
          select {
          case offer := <-s.OfferChan:
               s.ResponseChan <- s.acceptBid(offer)
          }
     }
}()
```

Remplacer l'appel à la fonction `acceptBid` par une communication utilisant les channels du vendeur pour une meilleur synchronisation concurrente.

STEP 5: Synchronization (line 220)

Vous n'avez rien à faire ici.

Le programme s'arrête lorsque les enchères se terminent, c'est-à-dire lorsqu'il n'y a plus d'acheteurs actifs ou lorsque toutes les propriétés sont vendues.