Méthodes et Types génériques

Exercice 1 : Exercice préliminaire
Récupérez l'archive fournie sur le portail.
Consultez le code et testez la classe `scanner.ScannerInt`. Vous utiliserez la méthode `saisieEntier` pour l'exercice suivant.

Exercice 2 : Génériques
Récupérez sur le portail l'archive contenant les fichiers en relation avec ce TP.

À chaque fois étudiez le "`main`" fourni pour orienter votre travail. Après avoir complété le code des classes tel que demandé dans les exercices, testez vos solutions, qui doivent compiler **sans aucun message d'avertissement** (« *warning* ») sur le typage.
Le "`main`" fourni doit fonctionner **sans aucune modification**.
Puis vous décommenterez les lignes précédées du commentaire `"NE COMPILE PAS"`, vous vérifierez alors que votre programme ne compile effectivement pas sans message et chercherez à comprendre pourquoi. S'il compile, c'est que votre réponse à la question est incorrecte !

Q 1 . Méthode générique

Ecrire la méthode `chose` de la classe `generics.ListChoser` qui :

- affiche un menu présentant la liste des valeurs des éléments de la liste en leur associant un numéro d'ordre, 0 correspondant à "aucun", 1 au premier élément de la liste, 2 au second, etc.
- demande de saisir une valeur entre 0 et le nombre d'éléments de la liste (inclus), et répète cette demande tant que la saisie est incorrecte (voir la classe `scanner.ScannerInt`),
- renvoie l'élément de la liste choisi (pas son indice !) ou `null` si 0 avait été choisi.

Dans cet exercice, la liste dans laquelle on choisit est évidemment typée, mais son type n'est pas connu initialement, et le type de retour de la méthode est du type des éléments de la liste. Il est donc nécessaire de rendre cette méthode **générique** (par rapport à ce type).

Testez votre solution à l'aide du code fourni dans le `main` de la classe `generics.ListChoser`.

Q 2 . "Contraintes" sur type générique

Q 2.1. Quelle **modification simple** faut il apporter à la méthode `chose` précédente pour qu'elle ne permette de choisir que dans une liste contenant des objets de type `Legume`, donc pas les `Pomme`.

Faites cette modification dans la classe `generics.ListChoserLegume` et testez la.

Q 2.2. Même question mais cette fois la méthode `chose` ne doit permettre que de choisir dans une liste contenant des objets de type `Legume` qui de plus sont `Cloneable`, donc pas les `Rutabaga`.

Faites la modification dans la classe `generics.ListChoserLegumeCloneable` et testez.

Q 3 . Type générique

On s'intéresse à la modélisation de "ramasseur/poseur". Ces êtres étranges ne peuvent ramasser ou déposer qu'un objet d'un seul type donné défini. On trouve ainsi des ramasseurs de carottes (et exclusivement de carottes) ou de choux-fleurs ou encore de pommes.

Ces êtres ne peuvent porter qu'un seul objet du type qu'ils acceptent à la fois. Les fonctionnalités d'un objet ramasseur sont :

▷ `prend` pour ramasser un objet, qui est alors passé en paramètre, à condition qu'il n'en possède encore aucun, sinon une `java.lang.IllegalStateException` est levée.

- ▷ `depose` pour déposer l'objet porté, il ne se passe rien si le ramasseur ne porte aucun objet, la méthode a pour résultat l'objet posé, `null` si aucun
- ▷ `donneA` pour donner l'objet porté à un autre ramasseur, qui doit être d'un type compatible bien sûr, il ne se passe rien si il ne porte pas d'objet et une exception `java.lang.IllegalStateException` est levée si le destinataire porte déjà un objet.
- ▷ il existe également un accesseur sur l'objet porté (dont le résultat sera `null` si aucun objet n'est porté).

Il faut donc ici créer un type `Ramasseur` paramétré par le type `T` des objets que le ramasseur peut ramasser, il aura un attribut de ce type et ses méthodes (les opérations qu'un ramasseur peut effectuer) dépendent de ce type.

Complétez la classe `Ramasseur` fournie, n'oubliez pas la javadoc, et faites les tests (ceux du fichier `Ramasseur.java` fourni et d'autres...).