Question 9 (**)

Nous souhaitons représenter une version simplifiée de la carte vitale où nous supposerons que son numéro n'est composé que de la concaténation du sexe (1 pour un homme, 2 pour une femme), de l'année de naissance de l'individu à quatre chiffres (1998 par exemple) et du mois de naissance de l'individu à deux chiffres. Par exemple, la coupe du monde de juillet 1998 aurait le numéro 2199807. Nous disposons des classes suivantes :

```
public class Carte{
  String numero;

  public Carte(String numero){
      this.numero=numero;
  }
}

public class Individu{
    boolean femme;
    int anneeNaissance;
    int moisNaissance;

    public Individu(boolean femme, int an, int mois){
      this.femme=femme;
      this.anneeNaissance=an;
      this.moisNaissance=mois;
    }
 }
```

Java dispose d'une interface fonctionnelle `Function<T,R>` qui n'a qu'une seule méthode `R apply(T t)`.

Déclarer une variable `nouvelIndividu` du bon type et l'initialiser à l'aide d'une expression lambda. Cette expression lambda doit décrire une fonction qui associe à une carte vitale (au numéro respectant la norme décrite ci-dessus) l'individu ayant cette carte. *(Vous pourrez utiliser les méthodes `Integer.parseInt(s)` pour convertir une chaîne de caractères `s` en entier, et `s.substring(i, j)` pour lire la sous-chaîne de `s` entre les indices `i` (inclus) et `j` (exclu).)*

```
Function<Carte,Individu> nouvelIndividu= (carte -> {
    boolean femme = true;
    if (carte.numero.charAt(0) == '1') femme = false;
    int anneeN = Integer.parseInt(carte.numero.substring(1,5));
    int moisN = Integer.parseInt(carte.numero.substring(6,7));
    return new Individu(femme, anneeN, moisN);
});
```

Question 11 (**)

```
public class B{}
public class A extends B{}
```

Réponses possibles :

- × `ArrayList<A>` est un sous-type de `ArrayList<B>`,
- → `ArrayList<B>` est un sous-type de `ArrayList<? super A>`,
- → `ArrayList<A>` est un sous-type de `ArrayList<? extends B>`
- → `ArrayList<A>` est un sous-type de `List<A>`.

Question 12 (*)

```
class B {
  int c = 0;
  public int getC () {
    return c;
  }
}
class A extends B {
  int c = 1;
}
public class Test {
  public static void main (String args[]) {
     A a = new A(); B b = a;
     System.out.print(b.getC()+" ");
     System.out.println(a.getC());
  }
}
```

Réponses possibles :

- × 1 1
- × 1 0
- × 0 1
- → 0 0

Question 13 (*)

```
public class B {
  public class A {}
}
```

Réponses possibles :

- × `B.A a = new A()`
- → `B.A a = b.new A()`, avec `b` une instance de `B`
- × `A a = b.new A()`, avec `b` une instance de `B`
- × `B.A a = new B.A()`

Question 14 (**)

Quelles affirmations suivantes sont correctes ?

Réponses possibles :

- × Une interface peut avoir une classe membre non-statique
- → Une classe abstraite peut contenir une méthode private
- × Une classe implémentant une interface doit implémenter/redéfinir toutes les méthodes déclarées dans l'interface
- → Dans une interface, toutes les paires possibles des modifieurs "private", "abstract" et "final" sont incompatibles dans une même déclaration de méthode.

pour la troisième pensez à default, ou a une classe abstraite implémentant l'interface Remarque : dans une classe, même abstraite, "private" et "final" ensemble sont ok (même si inutiles)

Question 10 (*)

Écrire une méthode statique qui prend en argument un tableau t et deux entiers i, j et qui écrase l'élément de t en position j avec l'element de t en position i. On ne suppose rien sur le type des éléments du tableau.

```
public static <T> void ecrase(T[] t, int i, int j) {
    t[j] = t[i];
}
```