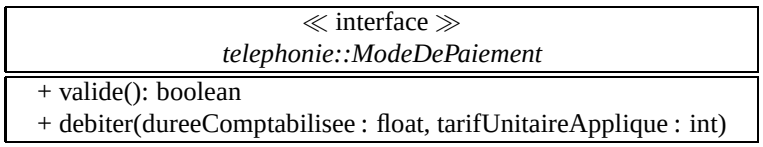
TD Téléphonie

Des fichiers utiles sont disponibles sur le portail.

Le but du problème est de simuler, à haut niveau, différentes fonctionnalités d'opérateurs de téléphonie et de modes de paiement d'utilisation de leur service.
Un *opérateur de téléphonie* doit permettre à un utilisateur d'ouvrir une *connexion* en fournissant un *mode de paiement* qui sera débité d'une certaine quantité en fonction de la politique tarifaire de l'opérateur. On décide de décrire ces trois entités par les interfaces et classes suivantes qui appartiennent toutes au paquetage `telephonie`.

```
public class Connexion {
  ...
  public Connexion(Operateur op, Date debut, ModeDePaiement m) { ... }
  public Operateur getOperateur() { ...}
  public void finConnexion(Date fin) { ... }
  public int heureDebutConnexion() { ... }
  public Date datedebutConnexion() { ... }
  public Date dateFinConnexion() { ... }
  public int dureeConnexion() { ... }            // en minutes, arrondi au sup
  public ModeDePaiement mode() { ... }
}
```

```
public interface Operateur {
  public Connexion seConnecter(Date debut,ModedePaiement m)
              throws OperateurSatureException, ModeDePaiementInvalideException ;
  public void seDeconnecter(Connexion c, Date fin) throws PasDeConnexionException ;
  public float getDureeComptabilisee(Connexion c);
  public int getTarifUnitaire(Connexion c);
}
```



Dans ce paquetage sont également définies les trois classes d'exception nécessaires :
`ModeDePaiementInvalideException`, `PasDeConnexionException` et `OperateurSatureException`.

Informations complémentaires.

- La méthode `seConnecter` d'un opérateur permet de débuter une conversation téléphonique en fournissant à celui-ci le mode de paiement qui sera débité lors de la déconnexion (appel à la méthode `seDeconnecter`).
- Un opérateur ne peut accepter qu'un nombre limité de connexions simultanées communiqué à sa construction; si ce nombre est atteint, une exception `OperateurSatureException` est levée. Il est donc nécessaire de mémoriser à la fois le nombre de connexions maximum et le nombre de connexions actives.
- Un opérateur refuse d'établir une connexion si le mode de paiement associé n'est pas valide[1] en levant une exception `ModeDePaiementInvalideException`.
- Lors d'une déconnexion, l'opérateur vérifie qu'il s'agit bien d'une connexion qui le concerne, sinon une exception `PasDeConnexionException` est levée. Dans l'autre cas, la fin de connexion est notifiée à la connexion (méthode `finConnexion`) et le mode de paiement associé à la connexion est débité en fonction de la durée de la connexion comptabilisée[2] exprimée en nombre entier de minutes[3] et du tarif unitaire appliqué[4] (coût d'une minute de communication) exprimé en nombre entier de centimes d'euros. Le débit est toujours effectué (voir comportement ci-dessous).

[1] méthode `valide()` de l'interface `ModeDePaiement`.
[2] paramètre `dureeComptabilisee` de la méthode `debiter` de l'interface `ModeDePaiement`.
[3] chaque minute commencée est comptabilisée et due à l'opérateur
[4] paramètre `tarifUnitaireApplique` de la méthode `debiter` de l'interface `ModeDePaiement`.

- On distingue deux types d'opérateurs en fonction de leur politique tarifaire correspondant à des calculs différents de la durée comptabilisée et du tarif unitaire appliqué :
  - les opérateurs à tarif fixe :

    Durée comptabilisée Un coefficient réducteur de 5/6 est appliqué sur la durée réelle de communication si cette dernière dépasse cinq minutes.

    Tarif unitaire appliqué Celui-ci est fixe et vaut 30 centimes d'euro par minute.
  - les opérateurs à tarif variable :

    Durée comptabilisée Il s'agit de la durée réelle de communication.

    Tarif unitaire appliqué Celui-ci est fonction de l'heure de début de connexion en suivant le tableau suivant :

Tarif unitaire	Heure de connexion h
15c	20h $\leq h <$ 8h
30c	8h $\leq h <$ 12h ou 14h $\leq h <$ 20h
45c	12h $\leq h <$ 14h

- Par ailleurs, il existe deux types de modes de paiement : les *cartes pré-payées* et les *cartes bancaires*. Chacun de ces modes de paiement implémente de façons différentes l'interface `ModeDePaiement`.

  1. les cartes bancaires sont toujours valides, le solde, exprimé en nombre entiers de euros, pouvant être négatif. Elles sont donc débitées d'une somme correspondant à la durée de communication comptabilisée multipliée par le tarif unitaire appliqué.
  2. les cartes pré-payées correspondent à un certain nombre de minutes de communications disponibles. Ce nombre de minutes ne peut jamais être négatif et une carte pré-payée est invalide dès que ce nombre est nul. Initialement, une carte pré-payée offre (si on peut dire) 50 minutes de communications au tarif unitaire de 15 centimes d'euros par minute. Il est nécessaire d'associer une carte bancaire à la construction de chaque carte pré-payée pour deux raisons :
     - la carte bancaire doit être immédiatement débitée du montnt correspondant au coût des 50 minutes de communication.
     - cette carte bancaire sert de caution en cas de dépassement de durée de communication comptabilisée. Si la durée de communication comptabilisée est inférieure au nombre de minutes restant sur la carte pré-payée, pas de problème, le nombre de minutes est diminué de cette durée. Dans le cas contraire, la durée restante, correspondant à la différence entre la durée de communication comptabilisée et le nombre de minutes encore disponibles sur la carte pré-payée, est facturée sur la carte bancaire associée à celle-ci au tarif unitaire propre à l'opérateur.

Vos classes devront se conformer, aux noms des méthodes près, à la simulation présentée à la question 4. Il est conseillée d'étudier les tests proposés par la classe `Simulation.java` avant de répondre aux questions, cela devrait pouvoir vous aider.

Q 1 . Donnez le code des classes permettant de modéliser les deux types d'opérateur (et donc conformes à l'interface `Opérateur`).

Q 2 . Donnez le code java de la classe `Connexion`. Pour gérer les heures de connexion et déconnexion on utilise la classe suivante :

telephonie::util::Date
...
+Date(...) +static getDateCourante() : Date *// la date au moment de l'invocation de la méthode* +getHeures() : int *// l'heure entre 0 et 23* +getMinutes() : int *// les minutes entre 0 et 59* + differenceEnMinutes(d:Date) : int *// ... "this-d" en mn entières (arrondi au sup)* +addMinutes(int mn) : Date

On supposera (sans vérification) qu'aucune connexion ne dure plus de 24h.

Q 3 . Donnez le code Java des entités permettant la modélisation des modes de paiement (il faut évidemment réutiliser le type `ModeDePaiement`).

Q 4 . *Simulation.* Afin de tester les différentes classes précédentes, vous pouvez utiliser la simulation proposée dans `Simulation.java`, éventuellement après avoir renommer certains identificateurs.

Cette simulation est définie selon les contraintes ci-dessous :

- créez un opérateur à tarif fixe acceptant au plus 3 connexions,
- créez un opérateur à tarif variable acceptant au plus 4 connexions,
- créez des modes de paiement : une carte bancaire (avec un solde très élevé) et une carte pré-payée associée à cette carte bancaire.
- essayez de créer 5 connexions sur chacun des opérateurs, ces connexions seront stockées dans des tableaux.

  Vous utiliserez les 2 modes de paiement créés précédemment.

  Les exceptions levées seront mises en évidence par un affichage.

  Les dates de début de connexion seront créées aléatoirement : jour identique pour toutes les connexions, heures entre 0 et 23 et minutes entre 0 et 59[5].
- déconnectez de son opérateur chacune des "premières" connexions de chaque opérateur. Les dates de fin de connexion seront créées aléatoirement en ajoutant un nombre de minutes aléatoire (et pas trop grand) à l'heure de début de connexion (méthode `addMinutes` de `Date`).

  Vous ajouterez dans les méthodes `debiter` des modes de paiement, une trace permettant de visualiser l'opération de débit.
- recréez pour chacun des opérateurs une nouvelle connexion,
- essayez de déconnecter une connexion du mauvais opérateur, l'exception levée sera mise en évidence par un affichage (rappel : on peut visualiser la pile des appels de mathodes au moment de l'exception par la méthode `printStackTrace()` de `Exception`).
- déconnectez toutes les connexions (qui ont été réellement créées), l'exception levée lors de l'invalidité d'un mode de paiement sera mise en évidence par un affichage.

[5] On rappelle qu'il existe principalement deux manières de produire des séquences aléatoires : la méthode statique `random()` de la classe `Math` et les méthodes `nextXXX()` de la classe `java.util.Random`