EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours Programmation II (SV)
Jamila Sam

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Série notée – Sujet 1

Instructions

— Max 75mn
— Max 80 points

1. Dans un navigateur, ouvrez la page
   se trouvant sous la rubrique **Séries d'exercices** de la page web du cours suivez scrupuleusement les instructions qui apparaissent.

2. Vous pouvez utiliser le navigateur une fois ces manipulations faites, pour consulter la page web du cours.

3. Pour **RENDRE** la série notée, en fin d'examen :
   (a) Allez à nouveau sur le lien donné précédemment et allez au point 3 des instructions. Le menu qui apparaît comporte déjà le nom du fichier à rendre.
   (b) Pour envoyer un fichier, cliquez sur le bouton `Browse` correspondant au nom du fichier et sélectionnez le fichier en question dans la liste qui vous est proposée. Cliquez ensuite sur le bouton `Envoyer`.
   Le fichier suivant est à envoyer, **même si non abordé** :

   cargaison.cc

   Pour terminer, n'oubliez pas de **VALIDER VOTRE ENVOI**.

Règles et recommandations :

1. La série est à réaliser **individuellement**. L'échange d'informations est strictement interdit (y compris l'échange de documents).

2. La série doit impérativement être réalisée sur les terminaux des salles CO020-023 et sur votre propre compte. Il ne doit y avoir **qu'un seul login sur votre compte durant le temps de la série notée**.

3. Vous pouvez amener votre propre clavier, mais il vous appartient de tester avant la série que ce matériel fonctionne correctement.

4. Il n'est pas permis : d'utiliser du matériel électronique, d'envoyer des emails, de vous connecter sur une autre machine ou d'imprimer pendant la durée de la série notée.

5. Documents autorisés: les documents du cours (transparents, séries, corrigés). Il vous est possible de consulter cette documentation, et seulement celle-là, en ligne. 1 livre de référence sur le langage C++, au plus 100 feuilles de notes personnelles. En cas de doute, demandez l'avis de l'assistant responsable. **Les corrigés d'anciens tests ne sont pas autorisés**.

6. Il est **impératif de mettre votre numéro SCIPER** en commentaire au début de chacun de vos fichiers.

7. Lisez attentivement chaque question de façon à ne faire que ce qui vous est demandé, et servez-vous de l'exemple de fonctionnement accompagnant les exercices pour vérifier votre solution. **Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.**

8. Si pour des raisons de langue, vous ne comprenez pas bien ce qui vous est demandé, n'hésitez pas à faire appel à un assistant pour obtenir des clarifications.

9. Sauf mention explicite, les affichages ne doivent pas obligatoirement correspondre à ceux donnés en exemple de fonctionnement; les mêmes informations doivent être affichées, mais le format peut être différent.

10. La série comporte un seul exercice.

11. Le fait que le code soit compilable ou exécutable compte dans la notation mais ne constitue pas le seul critère. La correction tiendra compte également des "solutions approchées".

Cargaisons chimiques (80 points)

On s'intéresse ici à «organiser» le transport de produits chimiques.

Vous prendrez note des directives suivantes avant de commencer à coder :

1. Vous travaillerez dans le fichier `cargaison.cc` et **n'utiliserez pas d'autres fichiers**. Ce fichier contient déjà :
   — la définition d'un type énuméré utile;
   — une fonction utilitaire `authorization` à décommenter le moment venu;
   — un programme principal auquel votre code devra strictement se conformer.
   les éléments fournis ne doivent pas être modifiés d'une autre façon que ce que l'énoncé vous suggère.
2. Vous rédigerez votre code en évitant la duplication inutile de code.
3. Vous compilerez sous la norme C++11. La commande dans le terminal est :

   ```
   g++ -std=c++11 cargaison.cc -o cargaison
   ```

4. Des endroits spécifiques sont prévus dans le fichier à compléter pour écrire votre code. **Codez aux endroits indiqués et il n'est pas nécessaire d'externaliser les définitions de méthodes**.

5. Vos solutions doivent utiliser le polymorphisme pour éviter les tests de type

6. Vous respecterez les principes d'une bonne encapsulation. Tous vos attributs seront notamment `private`

7. Si vous le jugez utile, vous êtes libres d'ajouter d'autres méthodes à celles qui vous seront explicitement demandées.

SUITE DE L'ÉNONCÉ au VERSO ⟶

Voici les éléments devant être modélisés :

1) Barils (POO de base, Héritage, Polymorphisme)

Il s'agit dans un premier temps de modéliser des barils destinés à transporter des produits chimiques.

Un *baril* standard (classe `Barrel`) est caractérisé par :
- le *nom* du produit qu'il contient, comme "acide chlorhydrique" par exemple;
- le *volume contenu* (un `double`);
- et le *ph* du produit contenu (un `double`).

Il existe par ailleurs deux catégories supplémentaires spécifiques de barils : ceux pour le transport de produits inflammables (la classe `Inflammable` sera utilisée pour réprésenter cette sorte de barils) et ceux pour le transport de substances corrosives (classe `Corrosive`). Les `Inflammable` seront caractérisés par le degré d'inflammabilité du produit transporté et les `Corrosive` par une information indiquant si le produit corrosif transporté est nocif pour l'environnement.
Codez une hiérarchie de classes permettant une utilisation polymorhique des barils et conforme au programme principal fourni entre `// TEST PARTIE 1` et `// FIN TEST PARTIE 1`.

Cette hiérarchie devra comporter :
- Des constructeurs initialisant les attributs au moyen de valeurs passées en paramètres et conformes au `main` fourni. Les constructeurs par défaut initialiseront le nom à *"Vide"*, le volume à zéro, le ph à 7.0 et, le cas échéant, le degré d'inflammabilité à `InflDegre::NUL` et la nocivité pour l'environnement à `false`. Pour simplifier, vous ne vous préoccuperez pas de la gestion des cas d'erreur (mauvaise valeurs pour le PH ou le volume par exemple).

- Une méthode `clear()` permettant de vider un baril : le volume sera remis à zéro,, le nom à *"Vide"*, le ph à 7.0 et, le cas échéant, le degré d'inflammabilité à `InflDegre::NUL` et la nocivité pour l'environnement à `false`.

- Une méthode `bool hazardous()` indiquant si le contenu du baril est dangereux. Cette méthode retournera toujours `false` pour un baril standard. Pour un `Inflammable`, `bool hazardous()` retournera `true` si le degré d'inflammabilité est fort et `false` autrement. Pour un `Corrosive`, elle retournera `true` si le produit transporté est nocif pour l'environnement et si le PH est strictement inférieur à 3 ou strictement supérieur à 11.

- une méthode `label(ostream&)` affichant, sur le flot passé en argument, le texte à afficher sur le baril lors de son transport. Ce label spécifiera si le produit transporté est inflammable ou corrosif, le nom du produit transporté, le volume et le ph ainsi que le texte `Transport Dangereux` si le contenu du baril dangereux et `Transport standard` s'il ne l'est pas. Il sera aussi indiqué, si un produit corrosif est nocif pour l'environnement. Ceci se traduira par l'affichage du texte : `Polluant`. Voici quelques exemples d'affichages :

  ```
  Corrosif, Polluant, Acide nitrique: 50 ml, PH=2, Transport Dangereux
  Corrosif, Ammoniaque: 100 ml, PH=11, Transport standard
  Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
  Chlorure d'hydrogène: 20 ml, PH=8, Transport standard
  ```

  Les deux premiers exemples correspondent à l'affichage d'un baril de type `Corrosive` (nocif et non nocif pour l'environnement), le troisième à celui d'un `Inflammable` et le dernier à celui d'un baril standard.

Vous doterez en outre la classe `Barrel` d'une surcharge de l'opérateur `operator-=(double vol)` permettant de soustraire au volume du baril la quantité `vol`. Le volume du baril ne peut pas devenir négatif suite à l'application de cet opérateur (voir l'exemple d'exécution ci-dessous). Si le baril devient vide. Le nom devient "Vide" et le PH 7.
La portion du programme principal comprise entre `// TEST PARTIE 1` et `// FIN TEST PARTIE 1`, une fois décommentée, doit produire l'affichage suivant:

```
Test partie 1 :
---------------
Vide: 0 ml, PH=7, Transport standard
Corrosif, Vide: 0 ml, PH=7, Transport standard
Inflammable, Vide: 0 ml, PH=7, Transport standard

un baril:
Chlorure d'hydrogène: 50 ml, PH=8, Transport standard
après retrait d'un volume de 30:
Chlorure d'hydrogène: 20 ml, PH=8, Transport standard
après encore un retrait d'un volume de 30:
Vide: 0 ml, PH=7, Transport standard

Corrosif, Ammoniaque: 100 ml, PH=11, Transport standard
Corrosif, Polluant, Acide nitrique: 50 ml, PH=2, Transport Dangereux
Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
Inflammable, Acétone: 300 ml, PH=7, Transport standard

un baril standard:
Chlorure d'hydrogène: 50 ml, PH=8, Transport standard
on le vide:
Vide: 0 ml, PH=7, Transport standard
un baril "Inflammable" :
Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
on le vide:
Inflammable, Vide: 0 ml, PH=7, Transport standard
un baril "Corrosif" :
Corrosif, Polluant, Acide nitrique: 50 ml, PH=2, Transport Dangereux
on le vide:
Corrosif, Vide: 0 ml, PH=7, Transport standard
```

SUITE DE L'ÉNONCÉ au VERSO ⟶

2) Cargaison (POO de base, collection hétérogènes)

On s'intéresse ici à modéliser une *cargaison* de produits chimiques (classe `Cargo`) sous la forme d'une «collection hétérogène» de barils.

Cette classe héritera d'une classe abstraite, `CheckedCargo`, imposant à ses sous-classes instanciables de contenir une méthode `bool check() const`.

La méthode `check` retournera `true` si la cargaison est conforme aux normes de sécurité en matière de transport et `false` sinon. Elle ne peut pas être définie de façon concrète dans la classe `CheckedCargo`.

La classe `Cargo` sera dotée :

- d'une méthode `add` permettant d'ajouter un baril à la cargaison. Cette méthode sera conforme au `main` fourni (une fois la portion de code entre `// TEST 2` et `// FIN TEST 2` décommentée);
- d'une méthode `isEmpty` retournant `true` si la cargaison est vide (aucun baril) et `false` sinon.
- d'une méthode `void clear()` permettant de vider la cargaison sans en détruire les barils;
- d'une méthode `void content (ostream& out)` affichant sur `out` le contenu d'une cargaison. Ceci se fera en affichant les «*label*» de chacun de ses barils ou le texte "vide" si la cargaison est vide, selon un format ressemblant à ceci :
  ```
  **  Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
  **  Corrosif, Ammoniaque: 100 ml, PH=11, Transport standard
  ```

Vous considérerez qu'un `Cargo` n'a pas la propriété des barils qu'il contient.

Un `Cargo` sera conforme au normes de sécurité si (strictement) moins de 50% de ses barils ont un contenu dangereux.

Pour tester le programme implémenté jusqu'ici, décommentez la fonction fournie `authorization` ainsi que la partie du programme principal comprise entre `// TEST PARTIE 2` et `// FIN TEST PARTIE 2`.

La trace d'exécution pour cette partie devrait donc ressembler à ceci :

```
Test partie 2 :
---------------
Cargaison 1:
-----------

Contenu initial de la cargaison :
**  Vide: 0 ml, PH=7, Transport standard
**  Corrosif, Ammoniaque: 100 ml, PH=11, Transport standard
Transport permis

Cargaison 2:
-----------
Contenu initial de la cargaison :
**  Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
**  Corrosif, Ammoniaque: 100 ml, PH=11, Transport standard
Transport interdit
Cargaison videe.
```

```
Cargaison 3:
-----------
Contenu initial de la cargaison :
**  Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
**  Inflammable, Acétone: 300 ml, PH=7, Transport standard
**  Corrosif, Polluant, Acide nitrique: 50 ml, PH=2, Transport Dangereux
Transport interdit
Cargaison videe.

Cargaison 4:
-----------
Contenu initial de la cargaison :
**  Inflammable, Acétylène: 200 ml, PH=7, Transport Dangereux
**  Inflammable, Acétone: 300 ml, PH=7, Transport standard
**  Corrosif, Ammoniaque: 100 ml, PH=11, Transport standard
Transport permis
```