Module R3.04 – TP06
Tests Unitaires avec Google Test
2 séances de 1h30

- *Téléchargez l'archive TP06.zip et décompressez-la dans votre répertoire R3-04*
- *Ouvrez dans CLion le projet TP06*
- ***Depuis CLion, pour charger Google Test dans le projet, faites un clic-droit sur le fichier CMakeLists.txt et sélectionnez « Reload CMake Project »***

Exercice 1. Tester Unitairement la classe Salarie – Cours 5, Chapitre 11

Au TP précédent, nous avons développé une classe **`Salarie`** avec les « spécifications » suivantes :

- Un salarié a un nom qui doit être une chaîne de caractères, non vide, ne comportant que des caractères alphabétiques ou le caractère "-", débutant et finissant impérativement par un caractère alphabétique.
- Un salarié doit posséder un numéro de sécurité sociale composé de 13 chiffres et commençant impérativement par 1 ou 2
- La rémunération mensuelle brute d'un salarié est un nombre réel compris entre le Smic et 500 fois le Smic.

Si le nom, le numéro de SS ou le salaire d'un salarié sont incorrects, des exceptions doivent être levées, respectivement :
`NomIncorrectException`, **`NumeroIncorrectException`** et **`SalaireIncorrectException`**

La classe possède également une méthode pour calculer l'impôt d'un salarié selon le barème suivant qui s'applique au revenu **annuel** (donc 12 mois de salaire mensuel) :

Revenu	< 6000 €	< 12 000 €	< 26 600 €	< 71 400 €	< 151 200 €	≥ 151 200 €
Taux Impôt	0,0 %	5,5 %	14 %	30 %	41%	50 %

Un stagiaire, sans doute pas assez rigoureux, a développé une classe **`Salarie`** avec (entre autres) les méthodes suivantes :

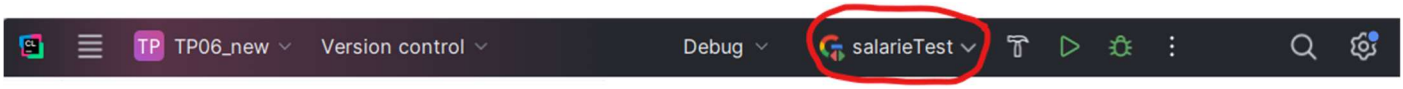
```
class Salarie {
public:
    Salarie(const std::string & nom = NOM_DEF,
            const std::string & numeroSS = NUMERO_SS_DEF,
            float salaireMensuel = SMIC);
    const std::string & getNom() const;
    void  setNom(const std::string & nom);
    void  setNumeroSS(const std::string & numeroSS);
    void  setSalaireMensuel(float salaireMensuel);
    float getImpot() const;
    // …
};
```

L'objectif du TP est d'écrire des tests unitaires, aussi exhaustifs que possibles, pour tester les méthodes de la classe **`Salarie`** énumérées ci-dessus et vérifier ainsi que les spécifications demandées ont bien été respectées. Il est très probable que le stagiaire a commis quelques erreurs en codant ces méthodes (il y en a 4 à trouver !). Vos tests doivent vous permettre, **sans regarder au préalable l'implémentation (approche « boîte noire »)**, d'identifier tous les problèmes !

Question 1. Tester la méthode getNom

Dans le fichier **`salarieTest.cpp`** (qui va contenir tous nos tests), complétez la fonction **`TEST(SalarieTest, GetNom)`** en écrivant des assertions qui devront vérifier que, pour un salarié construit avec différents noms valides, la méthode **`getNom`** renvoie bien toujours le bon nom, en majuscules.

Pour lancer vos tests, sélectionnez bien la cible « salarieTest » dans le menu « run » :



Si vos tests vous permettent d'identifier un problème dans la méthode **`getNom`**, corrigez-le jusqu'à ce que vos assertions soient toutes vérifiées.

Question 2. Tester la méthode setNom

Complétez la fonction **`TEST(SalarieTest, SetNom)`** pour tester la méthode **`setNom`**.
Vos assertions devront vérifier que la méthode lève bien la bonne exception pour différents noms invalides, ne lève pas d'exceptions pour différents noms valides et modifie bien le nom.
Corriger la méthode **`setNom`** si vous avez trouvé des problèmes

Question 3. Tester la méthode setNumeroSS

Complétez la fonction **`TEST(SalarieTest, SetNumeroSS)`** pour tester la méthode **`setNumeroSS`**.
Vos assertions devront vérifier que la méthode lève bien la bonne exception pour différents numéros invalides, ne lève pas d'exceptions pour différents numéros valides et modifie bien le numéro.
Corriger la méthode **`setNumeroSS`** si vous avez trouvé des problèmes

Question 4. Tester la méthode setSalaireMensuel

Complétez la fonction **`TEST(SalarieTest, SetSalaire)`** pour tester la méthode **`setSalaireMensuel`**.
Vos assertions devront vérifier que la méthode lève bien la bonne exception pour différents salaires invalides, ne lève pas d'exceptions pour différents salaires valides et modifie bien le salaire.
Corriger la méthode si vous avez trouvé des problèmes

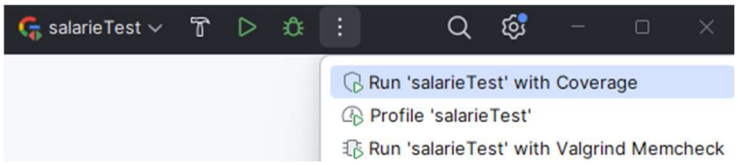
Question 5. Tester la méthode getImpot

Complétez la fonction **`TEST(SalarieTest, GetImpot)`** pour tester la méthode **`getImpot`**.
Vos assertions devront vérifier que la méthode calcule de façon correcte l'impôt d'un salarié, quelle que soit la tranche dans laquelle il se trouve. Vérifiez bien, en particulier, que le calcul est juste pour les revenus correspondant aux limites des tranches (utilisez la **`map TRANCHES_IMPOT`**).

Question 6. S'il vous reste du temps : tester le constructeur Salarie

Complétez la fonction **`TEST(SalarieTest, Salarie)`** pour tester le constructeur de la classe.
Comme vous travaillez en mode « boîte noire », vous ne pouvez pas faire l'hypothèse que ce constructeur a bien été codé en utilisant les setters **`setNom`**, **`setNumeroSS`** et **`setSalaire`**.
Vos assertions devront donc à nouveau vérifier que si on construit un **`Salarie`** avec des noms invalides, des numéro SS invalides ou des salaires invalides, les bonnes exceptions sont levées (et qu'elles ne le sont pas dans le cas d'informations valides).
Vous vérifierez également qu'un **`Salarie`** construit avec des informations valides contient bien, après construction, ces mêmes informations.

*A la fin du TP, vous pourrez lancer vos tests en mode « **`run with Coverage`** » pour savoir quel pourcentage du code de la classe **`Salarie`** est « couvert » par vos tests :*



Vous pourrez aussi faire tourner vos tests sur la classe Salarie que vous aviez écrite au TP05.