Séance 3: TESTS DU PROGRAMMEUR

Université Paris Cité

Objectifs:

- Tester des fonctions selon une spécification
- Utiliser les conditionnelles et les boucles
- Manipuler le type `int` des entiers
- Manipuler le type `String` des chaînes de caractères

L'une des grandes problématiques de la programmation informatique est le "bug", c'est-à-dire un programme qui ne termine pas ou qui ne renvoie pas le résultat attendu.

Le résultat attendu d'un programme est généralement spécifié sous la forme d'un *contrat* : il spécifie quelle est la forme des entrées que le programme attend et ce que le programme garantit alors sur ses sorties.

Il existe des techniques de preuve mathématique permettant de s'assurer qu'un programme respecte son contrat. Cependant, ces techniques ne sont pas au programme d'IP1.

Une autre méthode, moins sûre mais très répandue, permet de se convaincre de la correction du programme en faisant passer au programme un grand nombre de *tests*.

Un *test* est la donnée d'un couple formé d'une entrée I et d'une sortie attendue O respectant le contrat. Pour effectuer un test sur une fonction, on appelle cette fonction avec le paramètre I et on vérifie que la valeur de retour de la fonction est égale à la sortie attendue O définie par le test.

Important : les fonctions testées dans ce TP sont fournies mais il ne faut pas les modifier. Les tests sont en effet à programmer dans le `main`. En pratique, les fonctions ou les programmes testés ne sont pas toujours connus par le testeur. D'ailleurs, à la fin de ce TP, vous verrez comment en testant une fonction on peut arriver à comprendre ce qu'elle calcule.

Exercice 1 (Valeur Absolue, $\star$)
On rappelle que la valeur absolue $|x|$ *d'un entier* x *est sa valeur numérique considérée sans tenir compte du signe : la valeur absolue est toujours positive. On a donc* $|x| = x$ *si* $x \geq 0$*, et* $|x| = -x$ *si* $x < 0$*. Le fichier* `AbsoluteValue.java` *propose trois fonctions* `myAbs`, `sqrtAbs`, *et* `bitwiseAbs` *qui calculent la valeur absolue d'un entier donné en paramètre. Le but de cet exercice est de les tester. Les tests seront à placer dans la fonction* `main` *de ce fichier.*

1. *À l'aide de la procédure* `System.out.println`*, afficher dans le terminal la valeur de retour de la fonction* `myAbs` *évaluée sur l'entier* 0.
2. *On souhaite vérifier que la fonction* `myAbs` *vérifie le contrat suivant :*

   Contrat:

   x=-10 $\rightarrow$ *10*
   x=0 $\rightarrow$ *0*
   x=1 $\rightarrow$ *1*

Recopiez et modifiez la suite d'instructions :

```
System.out.println("Entrée : " + 0);
System.out.println("Sortie myAbs : " + myAbs(0));
```

pour afficher le résultat de chaque test de manière lisible :

```
Entrée : -10
Sortie myAbs : 10
Entrée : 0
Sortie myAbs : 0
Entrée : 1
Sortie myAbs : 1
```

3. *A l'aide d'une boucle, répétez ces tests pour toutes les entrées entre -10 et 10 inclus.*
4. *Java possède une fonction prédéfinie pour calculer une valeur absolue, appelée* `Math.abs`.
   À l'aide d'une conditionnelle, faites en sorte que votre affichage indique également si le résultat obtenu satisfait le contrat suivant :

   Contrat:
   Pour tout entier $-10 \leq i \leq 10$ *passé en paramètre,* `myAbs(i) == Math.abs(i)`.

   Par exemple, si myAbs renvoyait -1 pour l'entrée 1 (mais ce n'est pas le cas), on afficherait :

```
...
Entrée : -1
Sortie myAbs : 1 (valide)
Entrée : 0
Sortie myAbs : 0 (valide)
Entrée : 1
Sortie myAbs : -1 (erreur)
...
```

   Indication : *Pour afficher du texte sans retour à la ligne, on utilise la fonction* `System.out.print`, *plutôt que* `System.out.println`. *(Le "*`ln`*" à la fin du nom de la fonction signifie "line".)*
5. *Modifiez votre code pour tester aussi les fonctions* `sqrtAbs` *et* `bitwiseAbs`. *Tester tous les entiers entre -100 et 99 inclus à l'aide d'une seule et même boucle.*
6. *Testez maintenant tous les entiers entre -200 et 199 inclus. Qu'observe-t-on ? Modifiez votre code pour afficher uniquement les cas de test où une erreur est détectée.*
7. *On va maintenant effectuer des tests randomisés. La fonction* `randRange(a, b)` *qui vous est fournie renvoie aléatoirement un entier* `n` *compris entre les paramètres* `a` *et* `b` *:* $a \leq n < b$.
   Modifiez votre code de manière à tester les fonctions sur 100 entiers aléatoires compris entre -100 000 et 100 000. Parmi les fonctions `myAbs`, `sqrtAbs`, *et* `bitwiseAbs`, *lesquelles semblent correctes ?*
8. *Affichez la valeur de* `bitwiseAbs` *sur l'entrée 2 000 000. Qu'en conclure ?*

□

Exercice 2 (Palindrome, ⋆⋆)
Un mot est un palindrome *s'il se lit de la même manière de gauche à droite et de droite à gauche : par exemple « elle », « kayak » ou « ressasser » sont des palindromes. Le fichier* `Palindrome.Java` *contient trois fonctions* `isPalindrome1`, `isPalindrome2` *et* `isPalindrome3` *qui permettent de vérifier si un mot est un palindrome. Le but de cet exercice est de tester leur validité. Vous écrirez vos tests dans la fonction* `main` *de ce fichier.*

1. *On va d'abord tester la fonction* `reverse` *fournie, dont le but est d'inverser une chaîne de caractères. Affichez dans un terminal la valeur de retour de la fonction* `reverse`, *pour vérifier si elle satisfait le contrat suivant :*

Contrat:
À chaque paramètre `s` *est associé la valeur de retour attendue d'une fonction encodant l'inversion d'une chaîne de caractères :*

s = *"Miroir"*	→	*"rioriM"*
s = *"a"*	→	*"a"*
s = *""*	→	*""*
s = *"avec des espaces"*	→	*"secapse sed ceva"*
s = *"elle"*	→	*"elle"*

2. *Modifiez la fonction* `reverse` *pour corriger le bug observé à la question précédente.*
   Indice : *Rajouter un seul caractère suffit à rendre la fonction correcte.*
3. *Pour tester les fonctions sur de nombreux exemples, on va utiliser le dictionnaire de la langue française, fourni dans un fichier* `dico_fr.txt`. *La fonction* `wordFromDict` *prend en paramètre un entier* i *compris entre* 0 *et* 336530 *et renvoie la chaîne de caractères correspondant au* $i^{ème}$ *mot du dictionnaire. Comme à l'Exercice 1, on fournit également la fonction* `randRange(a, b)` *qui renvoie aléatoirement un entier compris entre les paramètres* `a` *et* `b`.
   À l'aide d'une boucle, générez une centaine d'entiers compris entre 0 *et* 336530, *et affichez les valeurs de retour des 3 fonctions* `isPalindrome` *sur quelques exemples choisis aléatoirement dans le dictionnaire.*
   Affichez les résultats sous la forme :

```
Entrée : hydrographique
Sortie isPalindrome1 : false
Sortie isPalindrome2 : false
Sortie isPalindrome3 : false
```

   Ce test est-il concluant ?
4. *On va maintenant tester* tous *les mots du dictionnaire. À l'aide d'une boucle et d'une conditionnelle, pour chacune des 3 fonctions* `isPalindrome`, *affichez tous les palindromes du dictionnaire, séparés par des virgules. Par exemple, votre affichage pourra ressembler à :*

```
Test de isPalindrome2 : a, à, alla, ana, ara, aviva, axa, bob, ...
```

   Laquelle des trois fonctions `isPalindrome` *est correcte ?*
5. *(⋆⋆⋆)* Optionnel : À faire tout à la fin du TP !
   Parfois, on considère qu'une phrase est un palindrome si elle contient les mêmes lettres dans les deux sens, ***sans prendre en compte*** *les espaces, la ponctuation, les majuscules, ni les accents. Des exemples de phrases palindromes :*
   — *Ésope reste ici et se repose.*
   — *Engage le jeu, que je le gagne !*
   Écrivez une fonction `isPalindromeSentence` *qui vérifie si une phrase est un palindrome.*
   Le fichier `perec.txt` *contient un texte de 1247 mots écrit par Georges Pérec, qui est un palindrome. La fonction* `loadPerec()` *renvoie une chaîne de caractères qui contient le texte complet. Utilisez votre fonction pour vérifier que c'est bien un palindrome.*

□

Exercice 3 (Mystère, ⋆⋆⋆)
Le fichier `Mystery.java` *contient* 8 *fonctions nommées* `mystere0`, `mystere1`,..., `mystere7` *qui prennent en paramètre soit un entier, soit une chaîne de caractères et renvoient, soit un entier, soit une chaîne de caractères. À l'aide de tests, déterminer leur comportement décrit sous forme d'un contrat le plus précis possible, puis vérifier votre hypothèse.* □