Introduction aux tests

DEV 2.3 (R2.03 Qualité de développement)

Florent Madelaine

IUT de Sénart Fontainebleau
Département Informatique

Année 2023-2024
Cours 1

Les tests c'est quoi ?

Domaine très vaste, qu'on va juste effleurer. On y reviendra au S3 dans le cadre du cours d'ACDA32 (M3301). Il y a des liens avec les bonnes pratiques de développement que vous voyez en ASR, APL, WIM et SGBD. Il y a des liens avec les méthodes formelles du cours MAT41 de Régine au S4 (M4105).

Idéal

On voudrait pouvoir savoir si un logiciel fait ce qu'il doit faire.

`https://en.wikipedia.org/wiki/Software_testing`

Au programme

Cette année

- Comprendre pourquoi c'est important de tester
- Appréhender les test unitaires.
- Voir le mécanisme d'assertion en java.
- Revoir les exceptions en java dans le cadre de la programmation défensive.
- Utilisation de JUnit pour faire des tests.
- Comprendre les limitations de ce qu'on peut tester.

L'an prochain

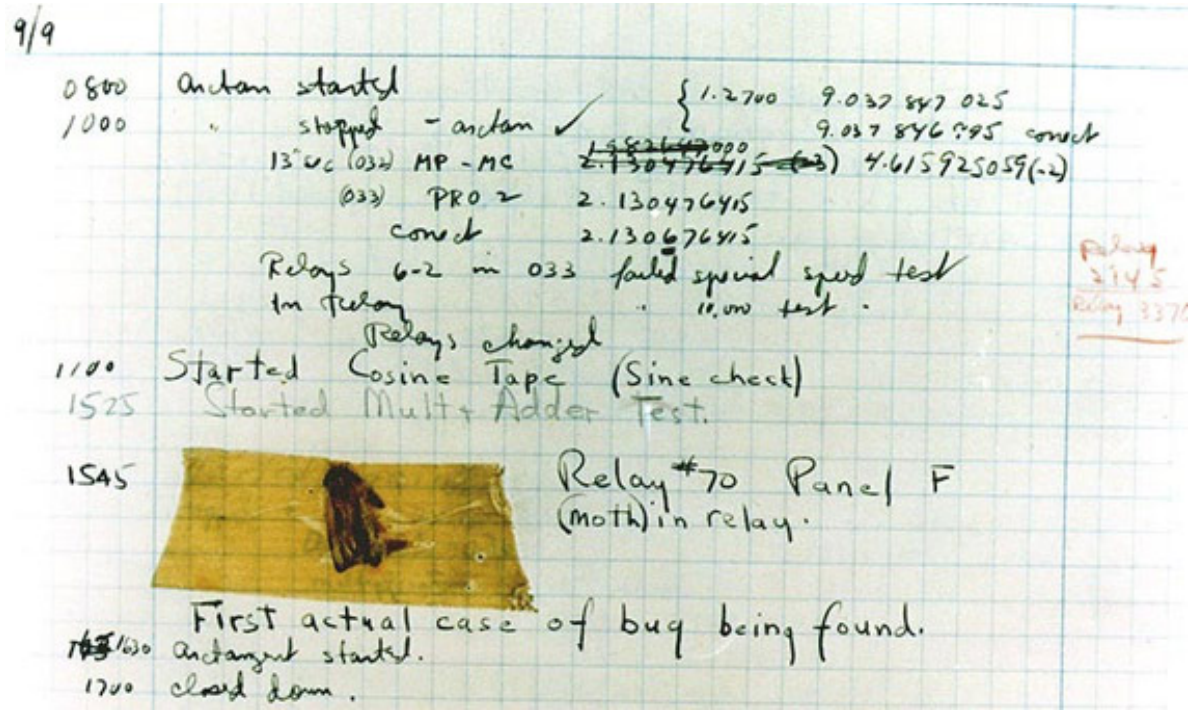
- Revoir l'utilisation de JUnit dans le cadre d'un projet
- Méthodologie du *Test-driven development*

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html https://junit.org/junit4/

Évaluation

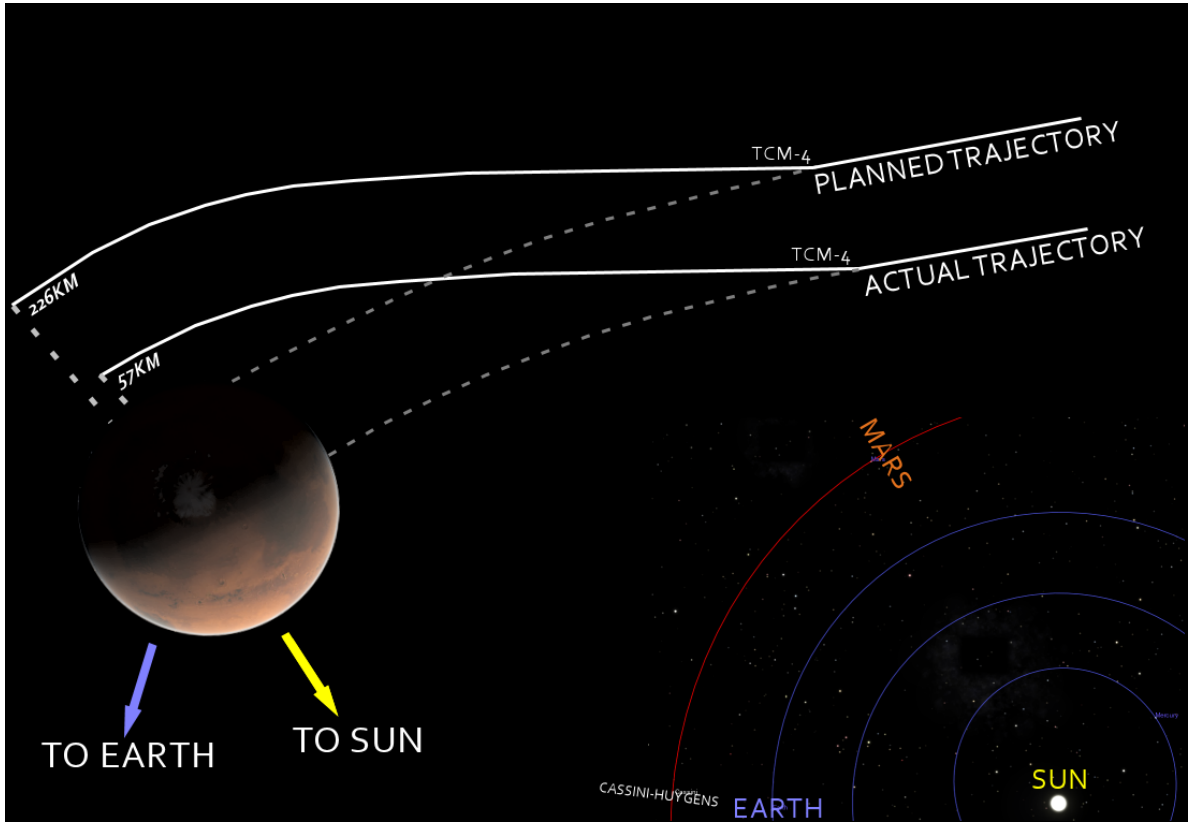
Concernant l'évaluation, nous vous recontacterons avec plus de détails : il s'agira d'un TP noté dans lequel il faudra compléter des tests sur un code existant (tester), et corriger du code existant à partir de tests donnés (debug).
Il conviendra donc de faire en sorte que vous puissiez travailler avec JUnit sur une machine du département.

Le premier bug (Mark II 1947)



https://en.wikipedia.org/wiki/Software_bug

Système métrique vs impérial (1998)



https://en.wikipedia.org/wiki/Mars_Climate_Orbiter

Problème de représentation

Il y a l'explosion d'Ariane 5 en 1996 liée à une conversion d'un float 64 bits vers un signed int 16 bits.

Moins triste et plus amusant, le bug suivant de Civilisation.

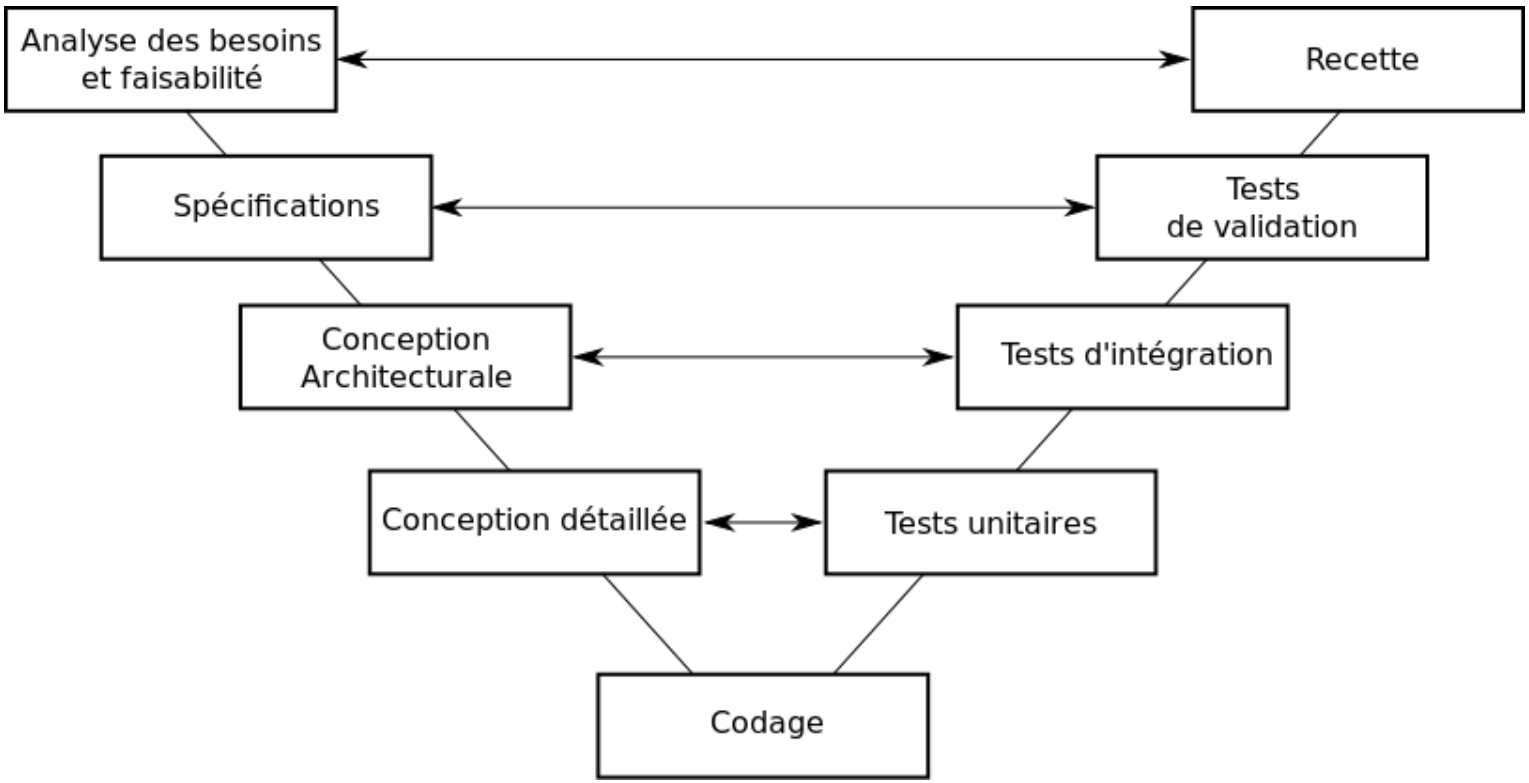
> *The first game in the Civilization series contained a notorious bug that caused one of the world leaders, Mahatma Gandhi, to behave like an aggressive warmonger, despite being known for advocating peace in the real world. The bug, which became famously known as "Nuclear Gandhi", became possible when Gandhi's aggression rating, represented as an 8-bit unsigned integer, was set to the lowest positive value of 1. If the player chose to democratize his native India, the rating would decrease by two, causing it to roll over back to the highest value, 255, thus making him the most aggressive leader in the game. The bug was so famous that the developers decided to allow players to deliberately goad Gandhi into aggressive conflict in later sequels.*

`https://en.wikipedia.org/wiki/List_of_software_bugs`

Place des tests dans le génie logiciel

Les tests sont de natures assez différentes et apparaissent tout au long du cycle de vie d'un logiciel.
Nous allons illustrer leur importance dans le cadre du cycle en V.

Cycle en V


Analyse des besoins et faisabilité
Recette
Spécifications
Tests de validation
Conception Architecturale
Tests d'intégration
Conception détaillée
Tests unitaires
Codage

Différents niveaux de tests

Dans le cycle en V on voit que des tests apparaissent à différents niveaux et selon l'organisation structurelle du code. Ceci est vrai en fait quelle que soit la méthode de développement. Ils portent par exemple

- sur des petits niveaux de détails du logiciel (tests unitaires)
- sur le bon agencement de briques logicielles (tests d'intégration)
- sur le système entier,
- jusqu'à la validation par des utilisateurs (recette)

Différents façons de tester

Il y a deux grandes familles de tests :

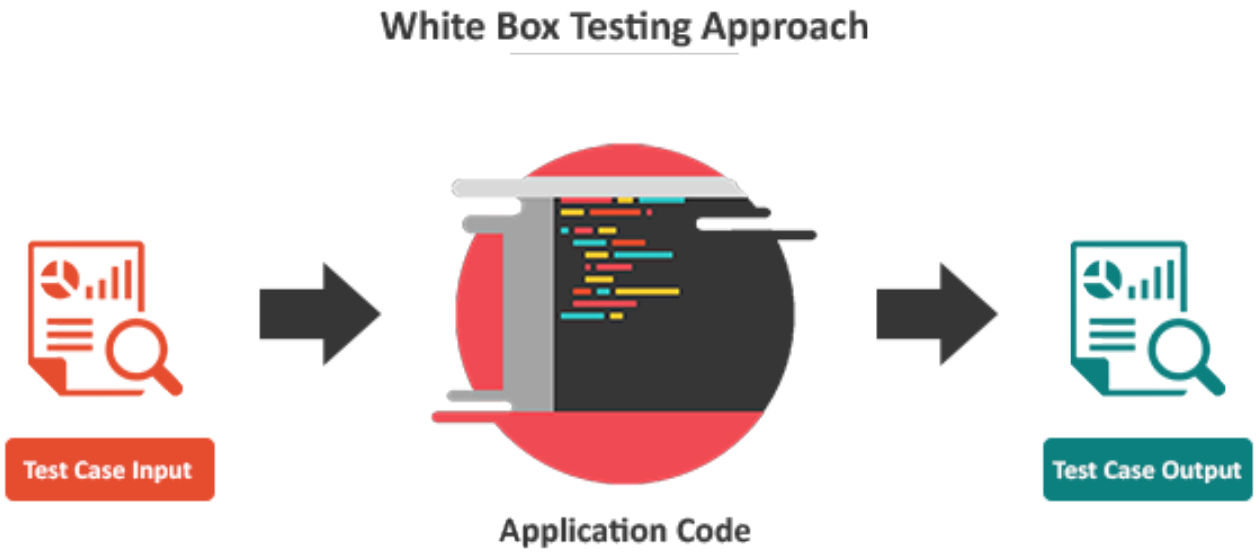
- boîte noire (on ne dispose pas des sources)
- boîte blanche (on dispose des sources)

Et une famille intermédiaire

- boîte grise (entre les deux).

Boîte blanche

Avec les tests unitaires, on a un exemple de test en boîte blanche dynamique (on interprète ou exécute le code sur un jeu de données).



Il y a aussi des formes de tests statiques (le code n'est pas exécuté). On en reparlera la semaine prochaine.

Tests unitaires

Typiquement, on prépare un test sur une petite partie du logiciel.

Avantages

- Test très simple donc probablement correct (et oui on peut se tromper aussi en écrivant des tests).
- Test très simple donc probablement automatisable (pas besoin de le lancer manuellement à l'avenir)
- On peut tester qu'il n'y a pas de régression entre 2 versions du logiciel.

Pat and Dale I

L'histoire suivante est extraite du livre *Pragmatic Unit Testing in Java 8 with JUnit* par Jeff Langr, Andy Hunt et Dave Thomas. C'est un pastiche du lièvre et de la tortue.

Pat and Dale II

Once upon a time–maybe it was last Tuesday– there were two developers, Pat and Dale. They were both up against the same deadline, which was rapidly approaching. Pat was pumping out code pretty fast; developing class after class and method after method, stopping every so often to make sure that the code would compile.

Pat kept up this pace right until the night before the deadline, when it would be time to demonstrate all this code. Pat ran the top-level program, but didn't get any output at all. Nothing. Time to step through using the debugger. Hmm. That can't be right, thought Pat. There's no way that this variable could be zero by now. So Pat stepped back through the code, trying to track down the history of this elusive problem.

Pat and Dale III

It was getting late now. That bug was found and fixed, but Pat found several more during the process. And still, there was no output at all. Pat couldn't understand why. It just didn't make any sense.

Dale, meanwhile, wasn't churning out code nearly as fast. Dale would write a new routine and a short test to go along with it. Nothing fancy, just a simple test to see if the routine just written actually did what it was supposed to do. It took a little longer to think of the test, and write it, but Dale refused to move on until the new routine could prove itself. Only then would Dale move up and write the next routine that called it, and so on.

Pat and Dale IV

Dale rarely used the debugger, if ever, and was somewhat puzzled at the picture of Pat, head in hands, muttering various evil-sounding curses at the computer with wide, bloodshot eyes staring at all those debugger windows.

The deadline came and went, and Pat didn't make it. Dale's code was integrated and ran almost perfectly. One little glitch came up, but it was pretty easy to see where the problem was. Dale fixed it in just a few minutes.

Pat and Dale V

Now comes the punch line : Dale and Pat are the same age, and have roughly the same coding skills and mental prowess. The only difference is that Dale believes very strongly in unit testing, and tests every newly-crafted method before relying on it or using it from other code.

Pat does not. Pat "knows" that the code should work as written, and doesn't bother to try it until most of the code has been written. But by then it's too late, and it becomes very hard to try to locate the source of bugs, or even determine what's working and what's not.

Tests unitaires pour nous en java

Un test unitaire c'est quelque chose de très simple qui s'applique à une toute petite partie du code et qui permet à un développeur de se convaincre que cette partie du code fait ce qu'il pense qu'elle doit faire.
En pratique :

- Pour chaque classe un ensemble de tests.
- Pour chaque méthode un ensemble de tests.

En pratique

Pour commencer on peut juste écrire des tests unitaires à la main dans le *main* de chaque classe.
On peut aussi utiliser le mécanisme des assertions en java

Assertion en java

Le mécanisme d'assertion en java permet de faire des tests simples directement dans le code. Par défaut ce mécanisme est désactivé et il n'a donc pas d'influence.

Si on souhaite que le mécanisme d'assertion soit activé (typiquement pour faire des tests quand on est en train de coder) alors il faut lancer java avec l'option `-ea`.
On peut aussi faire des choses plus fines en activant les assertions seulement pour des paquets spécifiques, et en désactivant les assertions pour certaines parties (avec l'option `-da`).

Exemple

En pratique on "appelle" assert sur une expression booléenne. Par exemple,

```
assert( False )
```

va retourner une erreur (plus précisément une `AssertionError`).

```
java.lang.Object
    java.lang.Throwable
        java.lang.Error
            java.lang.AssertionError
```

https://docs.oracle.com/javase/8/docs/technotes/guides/language/assert.html

Détails option d'activation des assertions

option -ea et -da (extrait de man java)

```
-enableassertions[:[packagename]...|:classname]
       -ea[:[packagename]...|:classname]
           Enables assertions. By default, assertions are disabled in all
           packages and classes.

           With no arguments, -enableassertions (-ea) enables assertions in
           all packages and classes. With the packagename argument ending in
           ..., the switch enables assertions in the specified package and any
           subpackages. If the argument is simply ..., then the switch enables
           assertions in the unnamed package in the current working directory.
           With the classname argument, the switch enables assertions in the
           specified class.

           The -enableassertions (-ea) option applies to all class loaders and
           to system classes (which do not have a class loader). There is one
           exception to this rule: if the option is provided with no
           arguments, then it does not apply to system classes. This makes it
           easy to enable assertions in all classes except for system classes.
           The -enablesystemassertions option provides a separate switch to
           enable assertions in all system classes.

           To explicitly disable assertions in specific packages or classes,
           use the -disableassertions (-da) option. If a single command
           contains multiple instances of these switches, then they are
           processed in order before loading any classes. For example, to run
           the MyClass application with assertions enabled only in package
           com.wombat.fruitbat (and any subpackages) but disabled in class
           com.wombat.fruitbat.Brickbat, use the following command:

               java -ea:com.wombat.fruitbat... -da:com.wombat.fruitbat.Brickbat MyClass
```

AssertionError c’est quoi ?

Ce n’est pas tout à fait une exception (voir APL 2.1).

`http://www.iut-fbleau.fr/sitebp/apl21/exceptions/`

Une erreur, c’est essentiellement comme une exception non vérifiée qu’on ne doit vraiment pas attraper (mais on peut quand même si on le souhaite vraiment).

`https://stackoverflow.com/questions/912334/differences-between-exception-and-error`

Throwable en java

```
java.lang.Throwable (implements java.io.Serializable)
    java.lang.Error
        java.lang.AssertionError
        java.lang.LinkageError
            ...
        java.lang.ThreadDeath
        java.lang.VirtualMachineError
            ...
            java.lang.StackOverflowError
            ...
    java.lang.Exception
        ...
        java.lang.ReflectiveOperationException
            java.lang.ClassNotFoundException
            ...
        java.lang.RuntimeException
            ...
            java.lang.IllegalArgumentException
            ...
            java.lang.IllegalStateException
            java.lang.IndexOutOfBoundsException
            ...
            java.lang.NullPointerException
            ...
```

https://docs.oracle.com/javase/8/docs/api/java/lang/package-tree.html

Que tester avec les assertions ?

- Vérifier les préconditions d'une méthode
- Vérifier les postconditions d'une méthode
- Vérifier des invariants

Des ? ? ?conditions, ça veut dire quoi ?

précondition ce qui doit être vrai avant l'appel de la méthode (conditions à vérifier sur les valeurs des attributs de l'instance et/ou les paramètres de la méthode).

postcondition ce qui doit être vrai après l'exécution du `return` de la méthode (retour dans le bloc de code l'ayant appelée).

Ce sont des notions essentielles de ce qu'on appelle la programmation par contrat une méthodologie de génie logicielle qui vise à limiter les bugs.

Préconditions

Exemple : vérifier les paramètres

- un paramètre n'a pas le droit d'être null

```
public Foo(Bar bar) {
     this.bar = Objects.requireNonNull(bar);
 }
```

- un paramètre doit être dans une certain plage

```
/**
* Sets the refresh rate.
*
* @param  rate refresh rate, in frames per second.
* @throws IllegalArgumentException if rate <= 0 or
*          rate > MAX_REFRESH_RATE.
*/
public void setRefreshRate(int rate) {
    // Enforce specified precondition in public method
    if (rate <= 0 || rate > MAX_REFRESH_RATE)
        throw new IllegalArgumentException("Illegal rate: " + rate);

    setRefreshInterval(1000/rate);
}
```

source : doc java

Postcondition

Exemple : vérifier au moins en partie que la méthode a respecté son contrat

- L'inverse calculé est bien l'inverse ($3*122=1 \mod 365$)

```
/**
     * Returns a BigInteger whose value is (this-1 mod m).
     *
     * @param  m the modulus.
     * @return this-1 mod m.
     * @throws ArithmeticException  m <= 0, or this BigInteger
     *         has no multiplicative inverse mod m (that is, this BigInteger
     *         is not relatively prime to m).
     */
    public BigInteger modInverse(BigInteger m) {
        if (m.signum <= 0)
          throw new ArithmeticException("Modulus not positive: " + m);
        if (!this.gcd(m).equals(ONE))
          throw new ArithmeticException(this + " not invertible mod " + m);

        ... // Do the computation

        assert this.multiply(result).mod(m).equals(ONE);
        return result;
    }
```

- ajouter un élément à une liste a augmenté la taille de la liste de 1.

source : doc java

C’est quoi un invariant I

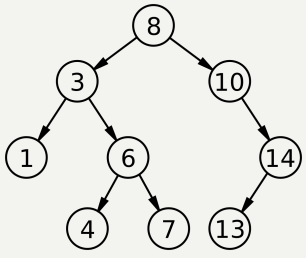
invariant une propriété qui reste vérifiée (pas forcément partout dans le code).

Exemples dans une liste

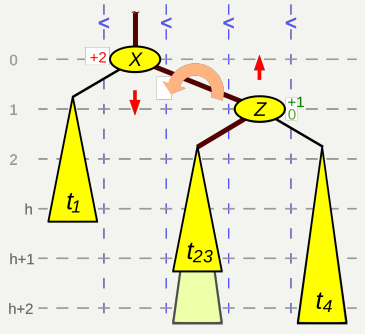
- déplacer un élément : le nombre d’éléments reste le même.
- trier une liste : le nombre d’éléments reste le même.

C’est quoi un invariant II

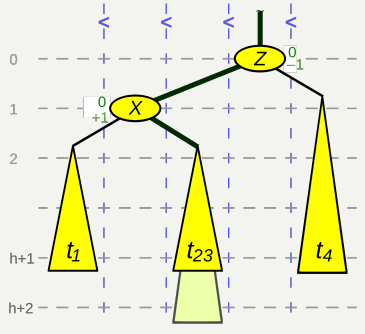
Exemple : arbre binaire équilibré



Les feuilles ont un écart de profondeur d’au plus 1.



Si une opération déséquilibre l’arbre (ajout, suppression), on utilise la technique de rotation pour rééquilibrer l’arbre



https://fr.wikipedia.org/wiki/Arbre_binaire_de_recherche

Que peut-on tester avec les assertions ?

Une précision

- Vérifier les préconditions d'une méthode privée
- Vérifier les postconditions d'une méthode
- Vérifier des invariants

Question

Pourquoi ne pas utiliser les assertions pour vérifier les préconditions des méthodes publiques ?

Limitation des assertion java

Comme les assertions java ne sont pas forcément activées, il est recommandé de ne pas les utiliser tout le temps.

En particulier, la documentation java recommande de ne pas utiliser le mécanisme des assertions pour les méthodes publiques d'une classe.

Programmation défensive en java

On bétonne les méthodes publiques pour éviter leur usage en dehors du cadre prescrit. On va toujours vérifier les préconditions et lever une exception adaptée si par exemple : un attribut est null et ne devrait pas l'être ; un attribut n'est pas dans la plage indiquée.
On peut aussi lever des exceptions quand lors de son exécution la méthode entraîne le système dans un état illégal (par exemple pour vérifier un invariant).

Préconditions paramètres d'une méthode et nouvelle classes

Remarque

On peut aussi déléguer à une nouvelle classe la vérification du type d'entrée d'une méthode, surtout si ce type se répète fréquemment dans notre code.
Par exemple, si on souhaite travailler souvent sur des entiers positifs ou nuls. Il suffit d'ajouter une classe PositiveInteger qui hérite d'Integer et de changer le paramètre de la méthode en PositiveInteger.