Mise en garde : fuite de référence

Ex :
- stockage dans un attribut public
- retour de méthode non privée
- passage en paramètre à une méthode d'une autre classe
- ...

Publier un objet :
rendre un objet accessible au-delà de l'endroit où il est censé être utilisé

Fuite de référence :
publication d'un objet mettant en péril le bon fonctionnement d'un programme

```
/** @inv getN() != 0 */
public class X {
    private int n;
    public X(final Y y) {
        n = 10;
        new Thread(new Runnable() {
            public void run() {
                y.m(this);
            }
        }).start();
    }
    public int getN() { return n; }
}
```

```
public class Y {
    public void m(X x) {
        System.out.println(x.getN());
    }
}
```

0 est un résultat possible !

```
public class X {
    private int n;
    Thread t;
    public X(final Y y) {
        n = 10;
        t = new Thread(new Runnable() {
            public void run() {
                y.m(this);
            }
        });
    }
    public void start() { t.start(); }
    public int getN() { return n; }
}
```

fuites de référence

X.this.getN()

```
/** @inv getN() != 0 */
public class X {
    private int n;
    public X(JButton source) {
        n = 10;
        source.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println(getN());
            }
        });
    }
    public int getN() { return n; }
}
```

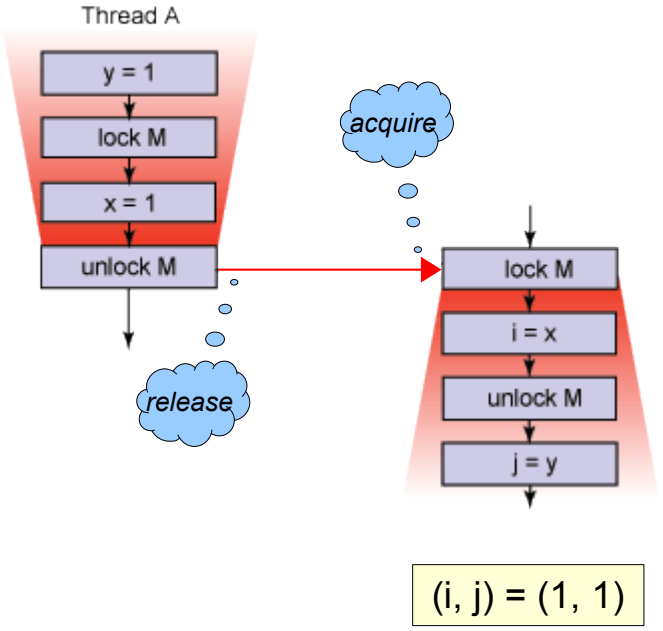
```
public class X {
    private int n;
    public X() { n = 10; }
    public void addBehaviour(JButton source) {
        source.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                System.out.println(getN());
            }
        });
    }
    public int getN() { return n; }
}
```

Règle méthodologique :
Ne jamais publier **this** ou une instance de classe interne depuis un constructeur

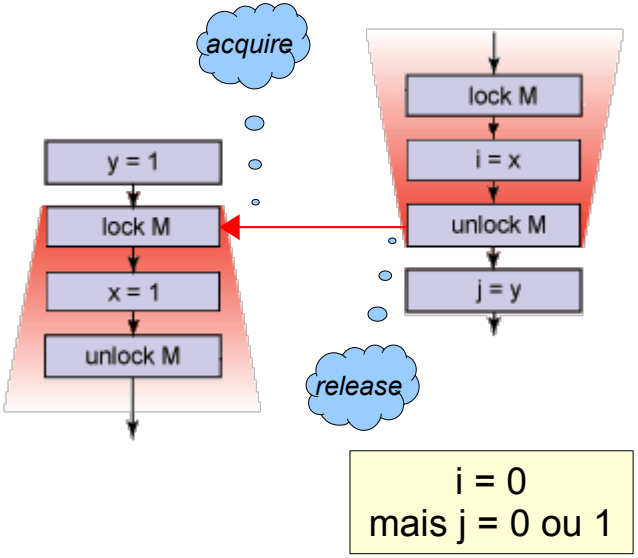
Mise en garde : accès compétitif aux données

(synchronisation inconsistante du code)

Accès compétitif aux données (*Data race*) :
plusieurs threads accèdent à la même variable partagée,
l'un d'eux au moins en écriture,
et il n'y a pas de relation d'antériorité entre les accès



(i, j) = (1, 1)



i = 0
mais j = 0 ou 1

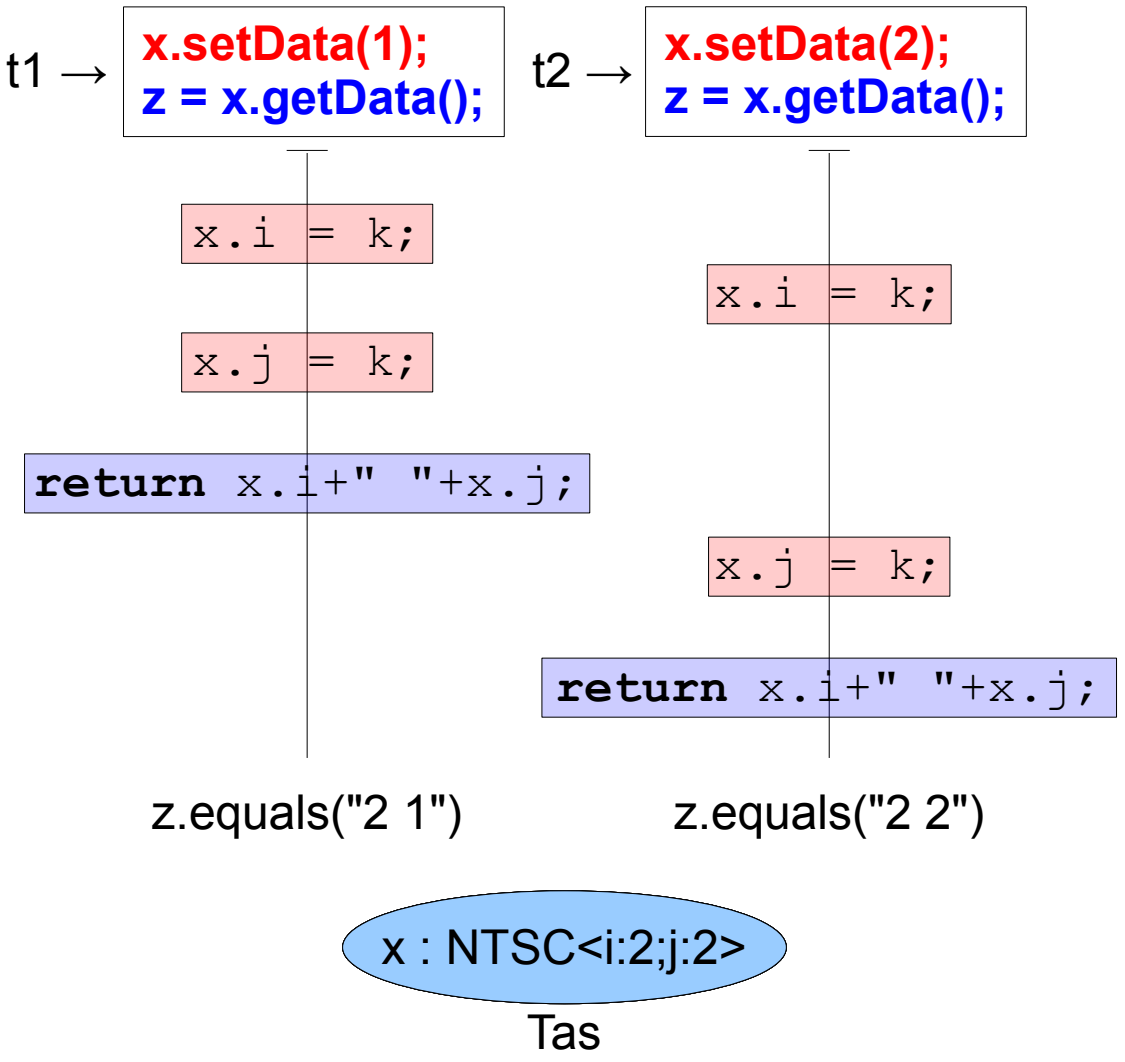
source de l'image : https://www.ibm.com/developerworks/library/j-jtp03304/

Thread-safety

- *Classe thread-safe* = chaque instance respecte son contrat, même dans un environnement concurrent
  - Ex. : `StringBuffer`, `Vector`...
  - Contre-ex. : `StringBuilder`, Swing...
- *Programme thread-safe* = respecte sa spécification, même dans un environnement concurrent

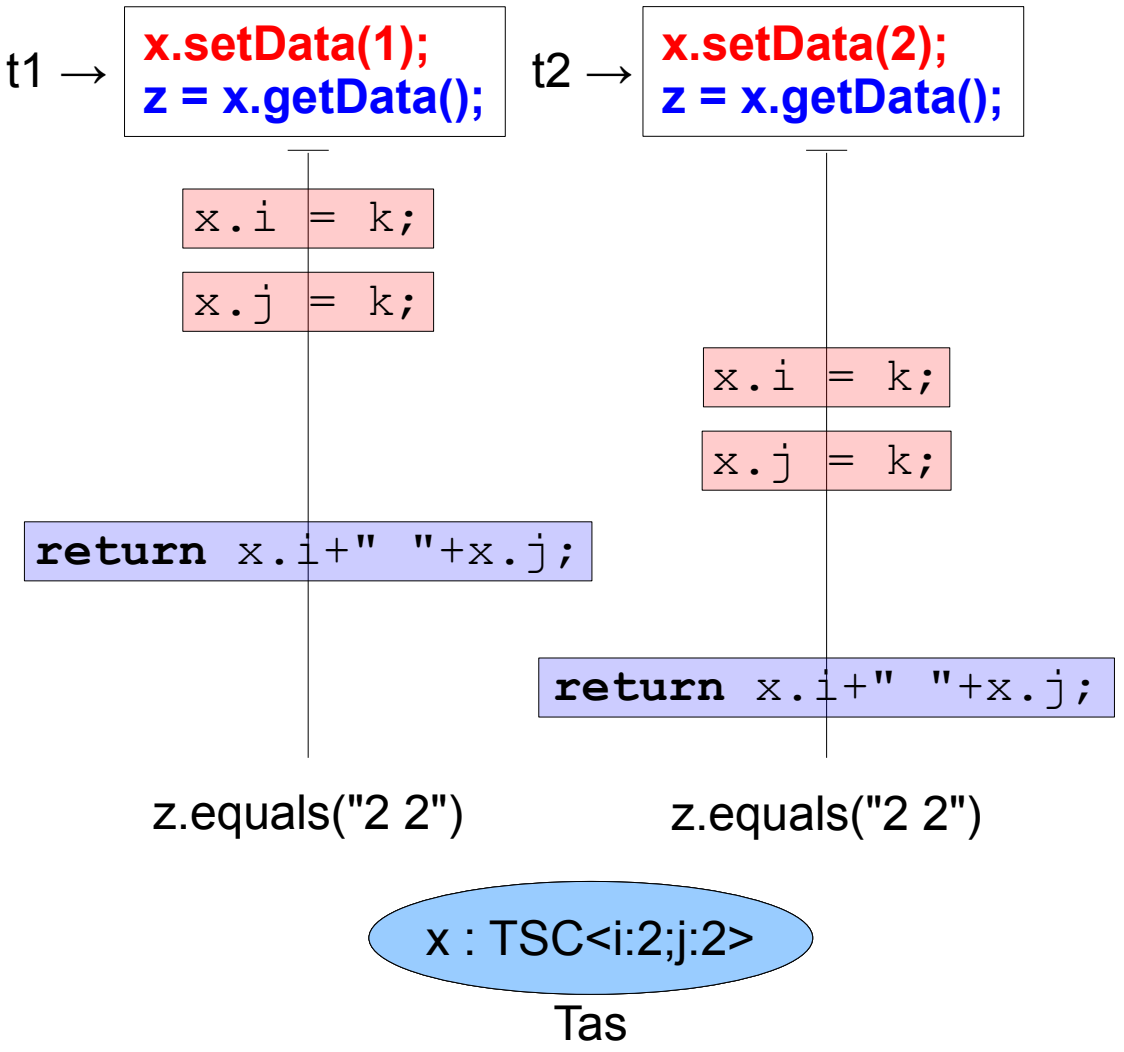
Exemple de classe non thread-safe

```
/**
 * @inv
 *      getI() == getJ()
 *      getData().equals(
 *          getI() + " " + getJ())
 */
class NonThreadSafeClass {
    private int i;
    private int j;
    int getI() {
        return i;
    }
    int getJ() {
        return j;
    }
    String getData() {
        return (i + " " + j);
    }
    /**
     * @post
     *      getI() == k
     */
    void setData(int k) {
        i = k;
        j = k;
    }
}
```



Exemple de classe thread-safe

```
/**
 * @inv
 *      getI() == getJ()
 *      getData().equals(
 *          getI() + " " + getJ())
 */
class ThreadSafeClass {
    private int i;
    private int j;
    synchronized int getI() {
        return i;
    }
    synchronized int getJ() {
        return j;
    }
    synchronized String getData() {
        return (i + " " + j);
    }
    /**
     * @post
     *      getI() == k
     */
    synchronized void setData(int k) {
        i = k;
        j = k;
    }
}
```



- Un programme thread-safe peut utiliser des classes non thread-safe

```
// spec : pour chacun des 5 threads
//        affiche deux fois le numéro d'ordre sur une ligne
class ThreadSafeProgram {
    public static void main(String[] args) {
        final NonThreadSafeClass x = new NonThreadSafeClass();
        for (int i = 0; i < 5; i++) {
            new Thread(new Runnable() {
                public void run() {
                    int n = getThreadNum(Thread.currentThread());
                    synchronized (x) {
                        x.setData(n);
                        System.out.println(x.getData());
                    }
                }
            }).start();
        }
    }
    ...
}
```

```
0 0
4 4
3 3
2 2
1 1
```

- Un programme n'utilisant que des classes thread-safe peut ne pas être thread-safe

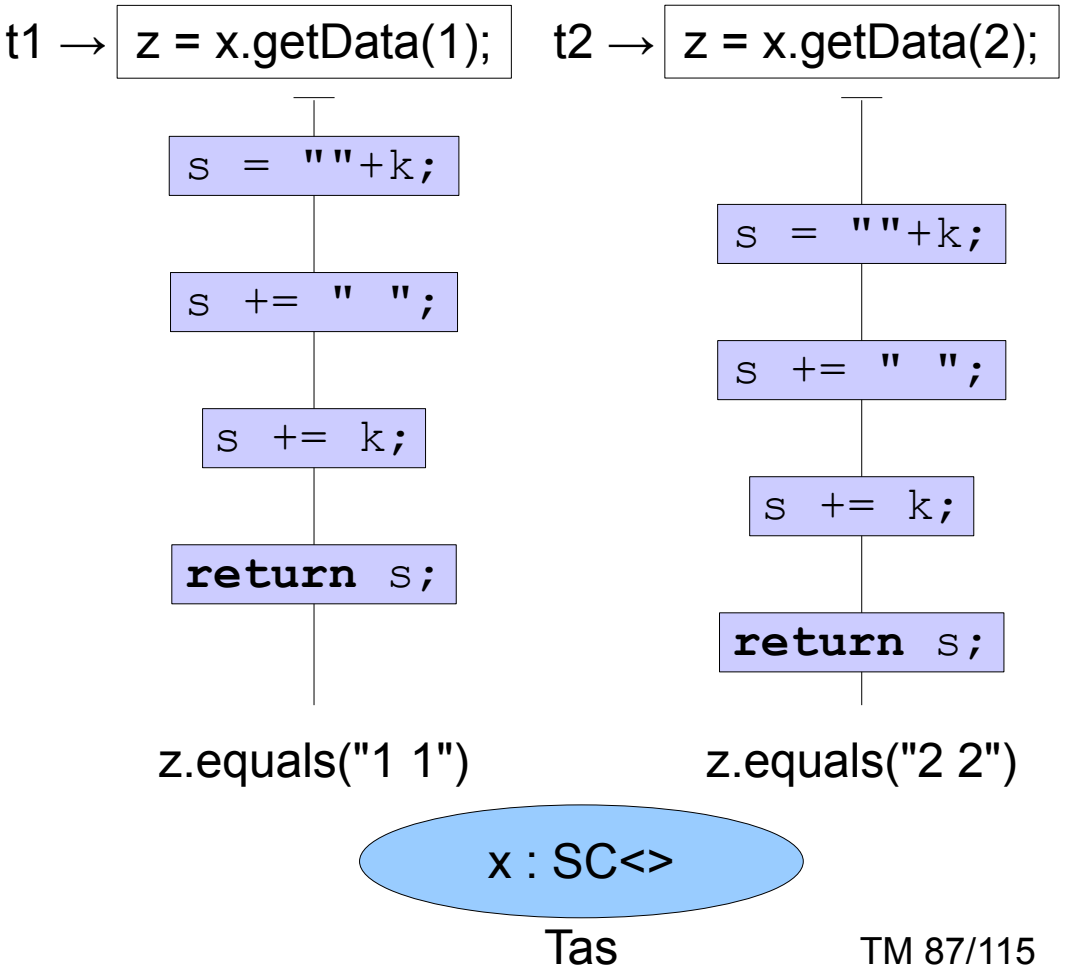
```
// spec : pour chacun des 5 threads
//        affiche deux fois le numéro d'ordre sur une ligne
class NonThreadSafeProgram {
    public static void main(String[] args) {
        final ThreadSafeClass x = new ThreadSafeClass();
        for (int i = 0; i < 5; i++) {
            new Thread(new Runnable() {
                public void run() {
                    int n = getThreadNum(Thread.currentThread());
                    x.setData(n);
                    System.out.println(x.getData());
                }
            }).start();
        }
    }
    ...
}
```

```
4 4
3 3
2 2
1 1
1 1
```

Règles de thread-safety

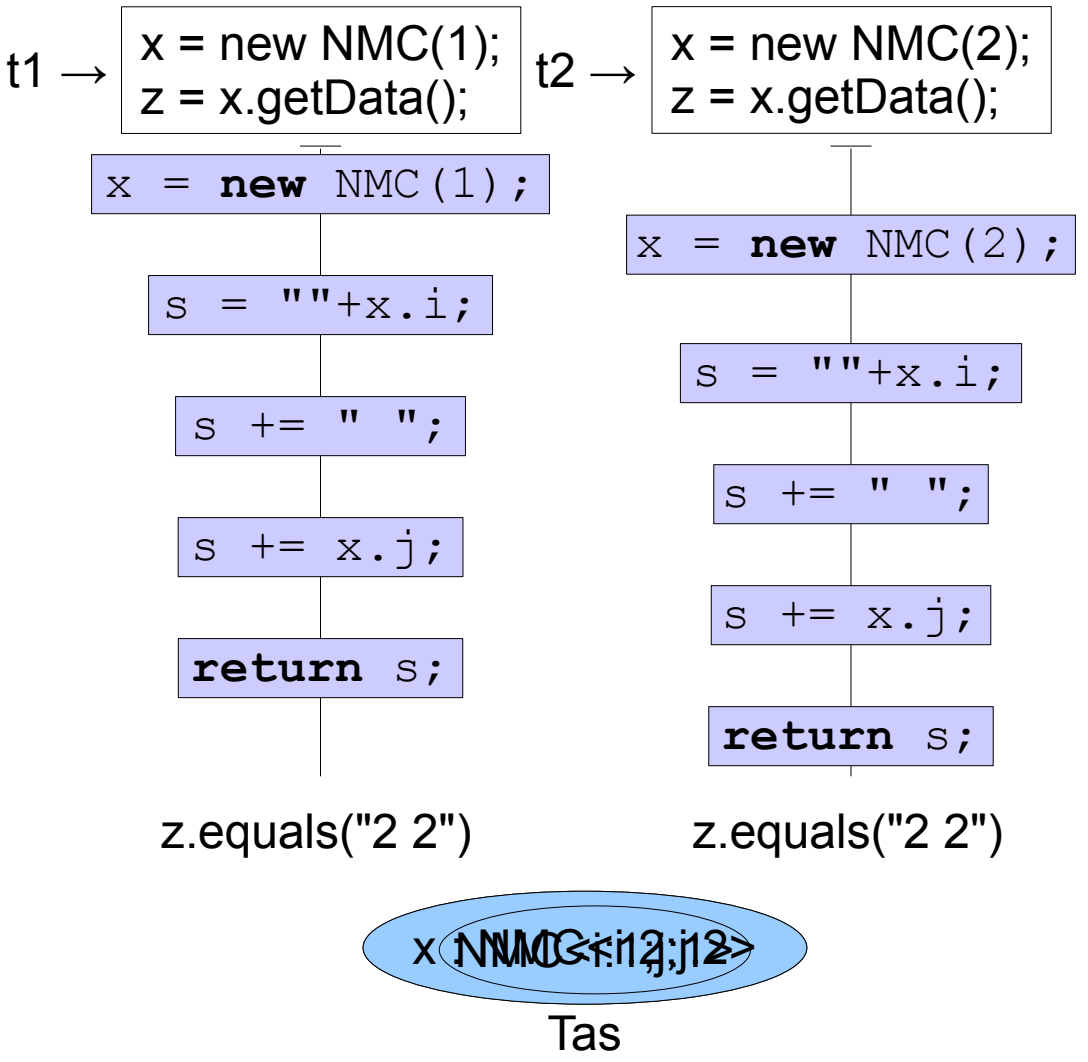
- R1 : Toute classe sans état est thread-safe

```
/**
 * @inv
 *      forall k:int
 *          getData(k).equals("k k")
 */
class StatelessClass {
    String getData(int k) {
        String s = String.valueOf(k);
        s += " ";
        s += k;
        return s;
    }
}
```



- R2 : Toute classe non mutable est thread-safe

```
/**
 * @inv
 *      getI() == getJ()
 *      getData().equals(
 *          getI() + " " + getJ())
 */
class NonMutableClass {
    private final int i;
    private final int j;
    /**
     * @post getI() == k
     */
    NonMutableClass(int k) {
        i = j = k;
    }
    int getI() {
        return i;
    }
    int getJ() {
        return j;
    }
    String getData() {
        String s = String.valueOf(i);
        delay();
        s += " ";
        delay();
        s += j;
        return s;
    }
}
```



- R3 : Tout attribut qui ne varie pas dans le code doit être déclaré **`final`**
  - seule l'observation des attributs **`final`** est garantie correcte après exécution du constructeur

```
class Test {
    static Test t;
    int x; final int y;
    Test() {
        x = 1;
        y = 2;
    }
    static void createT() {
        t = new Test();
    }
    static void printT() {
        if (t != null) {
            System.out.println("" + t.x + t.y);
        } else {
            System.out.println("null");
        }
    }
}
```

Si exécutées par deux threads distincts...

... alors l'affichage peut être
`null`, ou
`02`, ou
`12`

- R4 : Toute variable partagée, à mutation progressive* sur plusieurs threads, doit être *gardée* par un seul et même verrou
  - tout accès (lecture et écriture) à cette variable doit se trouver synchronisé sur cet unique verrou

```
class Accu {
    private final Object lock = new Object();
    // gardé par lock
    private long i;
    public void inc() {
        synchronized (lock) {
            i = i + 1;
        }
    }
    public long get() {
        synchronized (lock) {
            return i;
        }
    }
}
```

verrous identiques → aucun thread ne peut incrémenter i pendant qu'un autre en consulte la valeur

* mutation progressive :
le nouvel état est calculé à partir de l'ancien état
Ex : `x = x + 1;`

- R5 : Si plusieurs variables partagées sont liées entre elles par une relation d'invariance
  - *gardées* par un même verrou
  - modifiées par des *opérations atomiques*

```
class Pile {
    /* invariant de représentation
        data != null
        0 <= size <= data.length
        forall 0 ≤ i < size : data[i] != null
        forall size ≤ i < data.length : data[i] == null
     */
    // gardée par this
    private Integer[] data;
    // gardée par this
    private int size;
    Pile(int n) {
        synchronized (this) { data = new Integer[n]; }
    }
    synchronized Integer top() { ... }
    synchronized void pop() {
        if (size <= 0) throw new ISE();
        data[--size] = null;
    }
    synchronized void push(Integer x) { ... }
}
```

Un exemple complet : Le musée

- Un musée présente une exposition perpétuelle
- L'exposition est retransmise en direct par un cameraman qui filme, continuellement et en plan fixe, les tableaux de l'exposition
- Le directeur du musée change de temps en temps le style de son exposition en remplaçant les tableaux
- Le cameraman ne doit pas transmettre durant le changement des tableaux

Modélisation

- Les tableaux d'une exposition seront codés par une chaîne dont chaque caractère représente le style d'un tableau de l'exposition
- Ce que nous voulons observer, c'est le changement de style, pas les tableaux
  - style d'un tableau → une lettre
  - séquence des tableaux → une chaîne
  - changement de style → remplissage progressif de la chaîne avec une même lettre

- Voici ce que l'on aimerait obtenir :

```
 1 Voici la toute première exposition !
 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 3 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 4 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 5 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 6 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 7 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 8 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 9 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
11 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
14 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
15 ccccccccccccccccccccccccccccccccccccc
16 ccccccccccccccccccccccccccccccccccccc
17 ccccccccccccccccccccccccccccccccccccc
18 ddddddddddddddddddddddddddddddddddddd
19 ddddddddddddddddddddddddddddddddddddd
20 ddddddddddddddddddddddddddddddddddddd
```

Codage naïf (= sans synchro)

```
class Exposition {
    private static final String INITIAL_EXPO =
        "Voici la toute première exposition !";
    private int rang;
    private final StringBuilder tableaux;
    Exposition() {
        tableaux = new StringBuilder(INITIAL_EXPO);
    }
    String contenu() {
        rang += 1;
        return rang + " " + tableaux;
    }
    void changer(char style) {
        int n = tableaux.length();
        for (int i = 0; i < n; i++) {
            tableaux.setCharAt(i, style);
        }
    }
}
```

```
class Cameraman {
    private static final int MAX = 20;
    private final Exposition expo;
    private final Thread thread;
    Cameraman(Exposition e) {
        expo = e;
        thread = new Thread(new Runnable() {
            public void run() {
                // filmer
                for (int i = 0; i < MAX; i++) {
                    System.out.println(expo.contenu());
                }
            }
        });
    }
    void démarrer() {
        thread.start();
    }
}
```

```
class Directeur {
    private static final String STYLES =
        "abcdefghijklmnopqrstuvwxyz";
    private final Exposition expo;
    private final Thread thread;
    Directeur(Exposition e) {
        expo = e;
        thread = new Thread(new Runnable() {
            public void run() {
                // gérer le musée
                for (int i = 0; i < STYLES.length(); i++) {
                    expo.changer(STYLES.charAt(i));
                    attendreUnPeu();
                }
            }
        });
    }
    void démarrer() {
        thread.start();
    }
    ...
}
```

Résultat du codage naïf

```
class Test {
    public static void main(String[] args) {
        Exposition e = new Exposition();
        Cameraman c = new Cameraman(e);
        Directeur d = new Directeur(e);
        c.démarrer();
        d.démarrer();
    }
}
```

```
 1 Voici la toute première exposition !
 2 aaaaaala toute première exposition !
 3 aaaaaaaaaaaate première exposition !
 4 aaaaaaaaaaaaaaaaaamière exposition !
 5 aaaaaaaaaaaaaaaaaaaaaaaaexposition !
 6 aaaaaaaaaaaaaaaaaaaaaaaaaaaation !
 7 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 8 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 9 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
11 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
14 bbbbaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
15 bbbbbbbbbbaaaaaaaaaaaaaaaaaaaaaaaaa
16 bbbbbbbbbbbbbbbbaaaaaaaaaaaaaaaaaaa
17 bbbbbbbbbbbbbbbbbbbbbbaaaaaaaaaaaaa
18 bbbbbbbbbbbbbbbbbbbbbbbbbbbaaaaaaaa
19 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbaa
20 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
```

1- Pourquoi l'affichage des tableaux est-il ainsi ?

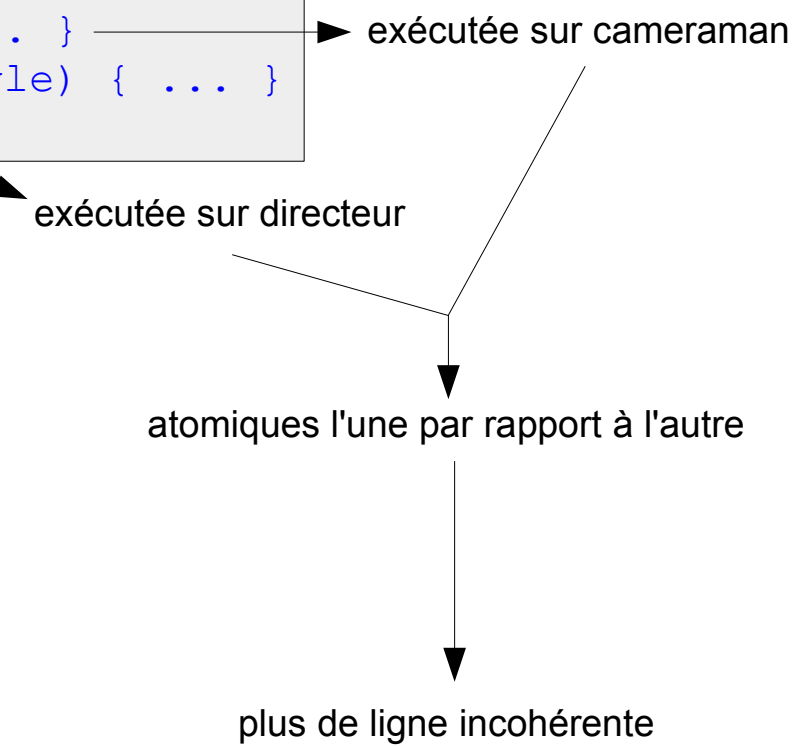
Le cameraman filme pendant que le directeur change les tableaux !

2- Pourquoi le rang est-il correct ?

La variable rang n'est accédée que par le thread du cameraman

- Solution 1 : rendre `Exposition` *thread-safe*

```
class Exposition {
    // gardée par this
    private final StringBuilder tableaux;
    ...
    synchronized String contenu() { ... }
    synchronized void changer(char style) { ... }
}
```




```
 1 Voici la toute première exposition !
 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 3 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 4 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 5 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 6 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 7 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 8 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 9 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
11 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
14 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
15 ccccccccccccccccccccccccccccccccccccc
16 ccccccccccccccccccccccccccccccccccccc
17 ddddddddddddddddddddddddddddddddddddd
18 ddddddddddddddddddddddddddddddddddddd
19 eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
20 eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
```

- Mais voici aussi ce que l'on pourrait obtenir :

```
 1 Voici la toute première exposition !
 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 3 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 4 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 5 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 6 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 7 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 8 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 9 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
10 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
11 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
12 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
13 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
14 ccccccccccccccccccccccccccccccccccccc
15 ccccccccccccccccccccccccccccccccccccc
16 ccccccccccccccccccccccccccccccccccccc
17 ddddddddddddddddddddddddddddddddddddd
18 ddddddddddddddddddddddddddddddddddddd
19 ddddddddddddddddddddddddddddddddddddd
20 ddddddddddddddddddddddddddddddddddddd
```

Observez la disparition des b

car il est possible que le cameraman n'ait pas pu filmer entre deux changements de style (nombreux processus, pas de chance, ...)

- Remarque : avec `StringBuffer`, on pourrait oser une synchronisation plus fine...

```
class Exposition {
    private static final String INITIAL_EXPO =
        "Voici la toute première exposition !";
    private int rang;
    // gardée par tableaux
    private final StringBuffer tableaux;
    Exposition() {
        tableaux = new StringBuffer(INITIAL_EXPO);
    }
    String contenu() {
        rang += 1;
        return rang + " " + tableaux;
    }
    void changer(char style) {
        int n = tableaux.length();
        synchronized (tableaux) {
            for (int i = 0; i < n; i++) {
                tableaux.setCharAt(i, style);
            }
        }
    }
}
```

`StringBuffer.toString`
`StringBuffer.setCharAt`
sont synchronisées

il faut synchroniser la boucle **for** pour qu'elle soit atomique !

- Solution 2 : synchroniser chez les clients

```
class Directeur {
    // gardée par expo
    private final Exposition expo;
    ...
        public void run() {
            // filmer
            for (int i = 0; i < STYLES.length(); i++) {
                synchronized (expo) {
                    expo.changer(STYLES.charAt(i));
                }
                attendreUnPeu();
            }
        }
    ...
}
```

```
class Cameraman {
    // gardée par expo
    private final Exposition expo;
    ...
        public void run() {
            // gérer le musée
            for (int i = 0; i < MAX; i++) {
                synchronized (expo) {
                    System.out.println(expo.contenu());
                }
            }
        }
    ...
}
```

`Exposition` est implantée de manière non *thread-safe* et à l'aide d'un `StringBuilder`

- Solution 1
  - si environnement non concurrent → coûteux
- Solution 2
  - clients doivent synchroniser les sections critiques (accès à l'état de l'exposition)
- Solutions non équivalentes
  - solution 1 : insuffisante en général
  - solution 2 : à préférer

Conditions

- Modifions l'énoncé :
  - cameraman → photographe
  - une seule photo par style d'exposition
  - attente du photographe entre deux styles différents

Algorithmes

- Exclusion mutuelle entre les corps des boucles

Photographe : photographier	**Directeur : gérer le musée**
Faire un certain nb de fois afficher les tableaux envoyer un signal à d attendre le signal de d Fait	Pour chaque style Faire changer le style envoyer un signal à p attendre le signal de p FinPour

Deadlock !

```
class Photographe {
    private final Object rendezVous;
    ...
        public void run() {
            // photographier
            for (int i = 0; i < MAX; i++) {
                synchronized (expo) {
                    System.out.println(expo.contenu());
                    synchronized (rendezVous) {
                        rendezVous.notify();
                        try {
                            rendezVous.wait();
                        } catch (IE e) {
                            // rien on continue
                        }
                    }
                }
            }
        }
    ...
}
```

```
class Directeur {
    private final Object rendezVous;
    ...
        public void run() {
            // gérer le musée
            for (int i = 0; i < STYLES.length(); i++) {
                synchronized (expo) {
                    expo.changer(STYLES.charAt(i));
                    synchronized (rendezVous) {
                        rendezVous.notify();
                        try {
                            rendezVous.wait();
                        } catch (IE e) {
                            // rien on continue
                        }
                    }
                }
            }
        }
    ...
}
```

Ces deux variables référencent le même objet via les constructeurs

Attente infinie... dommage !

Le seul pb est la terminaison du thread survivant

```
class Photographe {
    ...
        public void run() {
            // photographier
            for (int i = 0; i < MAX; i++) {
                synchronized (expo) {
                    System.out.println(expo.contenu());
                    expo.notify();
                    try {
                        expo.wait();
                    } catch (IE e) {
                        // rien on continue
                    }
                }
            }
        }
    ...
}
```

```
class Directeur {
    ...
        public void run() {
            // gérer le musée
            for (int i = 0; i < STYLES.length(); i++) {
                synchronized (expo) {
                    expo.changer(STYLES.charAt(i));
                    expo.notify();
                    try {
                        expo.wait();
                    } catch (IE e) {
                        // rien on continue
                    }
                }
            }
        }
    ...
}
```

Rq : dans ce contexte un *spurious wakeup* ou une interruption ne poseront pas de pb...

Voir le thread effaceur du TP 6 !

- Avec prise en compte de la terminaison :

```
 1 Voici la toute première exposition !
 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 3 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
 4 ccccccccccccccccccccccccccccccccccccc
 5 ddddddddddddddddddddddddddddddddddddd
 6 eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee
 7 fffffffffffffffffffffffffffffffffffff
 8 ggggggggggggggggggggggggggggggggggggg
 9 hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh
10 iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii
11 jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj
12 kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk
13 lllllllllllllllllllllllllllllllllllll
14 mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
15 nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn
16 ooooooooooooooooooooooooooooooooooooo
17 ppppppppppppppppppppppppppppppppppppp
18 qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq
19 rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr
20 sssssssssssssssssssssssssssssssssssss
```

Changement d'environnement

- Modifions encore un peu l'énoncé :
  - plusieurs photographes
  - mais un seul photographe est publié
    - dès qu'un photographe prend la photo, les autres abandonnent → ils n'imprimeront pas `Exposition.contenu()`
  - toujours un seul directeur qui attend un peu entre les changements de style
  - il n'y a plus de rendez-vous : chacun essaye d'obtenir le verrou pour travailler (en boucle)

```
class Photographe {
    private static String photo = "";
    Photographe(Exposition e, String name) {
        expo = e;
        thread = new Thread(new Runnable() {
            public void run() {
                // photographier
                for (int i = 0; i < MAX; i++) {
                    synchronized (expo) {
                        String data = expo.contenu();
                        if (!photo.equals(data)) {
                            photo = data;
                            System.out.println(photo + thread.getName());
                        }
                    }
                }
            }
        }, name);
    }
    ...
}
```

plusieurs instances de Photographe pour un seul Directeur

```
class Directeur {
    Directeur(Exposition e) {
        expo = e;
        thread = new Thread(new Runnable() {
            public void run() {
                // gérer le musée
                for (int i = 0; i < STYLES.length(); i++) {
                    synchronized (expo) {
                        expo.changer(STYLES.charAt(i));
                    }
                    attendreUnPeu();
                }
            }
        });
    }
    ...
}
```

```
class Test {
    public static void main(String[] args) {
        final int n = 5;
        Exposition e = new Exposition();
        Photographe[] ps = new Photographe[n];
        for (int i = 0; i < ps.length; i++) {
            ps[i] = new Photographe(e, "p" + (i + 1));
            ps[i].demarrer();
        }
        Directeur d = new Directeur(e);
        d.demarrer();
    }
}
```

Chaque photographe fait MAX tentatives
Certaines tentatives ne sont pas des succès
Il peut donc y avoir beaucoup moins de photos
(surtout si le directeur est très lent)

```
 1 Voici la toute première exposition ! par p1
 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa par p5
 3 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb par p1
 4 ccccccccccccccccccccccccccccccccccc par p4
 5 ddddddddddddddddddddddddddddddddddd par p1
 6 eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee par p4
 7 fffffffffffffffffffffffffffffffffff par p2
 8 ggggggggggggggggggggggggggggggggggg par p2
 9 hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh par p1
10 iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii par p1
```

- Il est possible d'utiliser une barrière du paquetage `java.util.concurrent`
  - tous les photographes sont bloqués sur la barrière
  - quand tout le monde est là, la barrière se lève
  - tous se ruent pour prendre une photo

volatile photoPrise serait-ce suffisant ?

```
class Photographe {
    private static boolean photoPrise = false;
    private final Exposition expo;
    private final Thread thread;
    private final Directeur dirlo;
    Photographe(Exposition e, String name, Directeur d) {
        expo = e;
        dirlo = d;
        thread = new Thread(new Runnable() {
            public void run() {
                while (true) {
                    synchronized (Photographe.class) {
                        if (!photoPrise) {
                            photoPrise = true;
                            System.out.println(
                                expo.contenu() + " par " + thread.getName()
                            );
                        }
                    }
                    try {
                        dirlo.getBarriere().await();
                    } catch (Exception e) {
                        // rien
                    }
                }
            }
        }, name);
    }
    void demarrer() { thread.start(); }
    static synchronized void photoAPrendre() { photoPrise = false; }
}
```

```
class Directeur {
    private static final String STYLES = "abcdefghijklmnopqrstuvwxyz";
    private final Exposition expo;
    private final CyclicBarrier barriere;
    Directeur(Exposition e, int nbPhotographes) {
        expo = e;
        barriere = new CyclicBarrier(
            nbPhotographes,
            new Runnable() {
                private int cpt;
                public void run() {
                    if (cpt < STYLES.length()) {
                        expo.changer(STYLES.charAt(cpt));
                        Photographe.photoAPrendre();
                        cpt += 1;
                    } else {
                        System.exit(0);
                    }
                }
            }
        );
    }
    CyclicBarrier getBarriere() {
        return barriere;
    }
}
```

Optionnel :
Tâche à exécuter chaque fois avant de relâcher les threads

Rq. le dirlo n'a plus besoin de démarrer

- Résultat :

```
class Test {
    public static void main(String[] args) {
        final int n = 5;
        Exposition e = new Exposition();
        Directeur d = new Directeur(e, n);
        Photographe[] ps = new Photographe[n];
        for (int i = 0; i < ps.length; i++) {
            ps[i] = new Photographe(e, "p" + (i + 1), d);
            ps[i].demarrer();
        }
    }
}
```

```
 1 Voici la toute première exposition ! par p1
 2 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa par p5
 3 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb par p1
 4 ccccccccccccccccccccccccccccccccccccc par p5
 5 ddddddddddddddddddddddddddddddddddddd par p1
 6 eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee par p5
 7 fffffffffffffffffffffffffffffffffffff par p1
 8 ggggggggggggggggggggggggggggggggggggg par p5
 9 hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh par p1
10 iiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiiii par p5
11 jjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjjj par p1
12 kkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkkk par p1
13 lllllllllllllllllllllllllllllllllllll par p2
14 mmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm par p1
15 nnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnnn par p2
16 ooooooooooooooooooooooooooooooooooooo par p1
17 pppppppppppppppppppppppppppppppppppp par p2
18 qqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqqq par p1
19 rrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrrr par p2
20 sssssssssssssssssssssssssssssssssssss par p1
21 ttttttttttttttttttttttttttttttttttttt par p2
22 uuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuuu par p1
23 vvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvvv par p2
24 wwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww par p1
25 xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx par p2
26 yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy par p1
27 zzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzzz par p2
```

Les photographes reviennent toujours à la charge.
Lorsque le directeur a passé tous les styles, le programme se termine.