

Algorithmes et Programmation : du séquentiel au distribué

Cours INF431, Promotion X2008

B. Werner J.-M. Steyaert

- Gr. 1, 6** : F. Magniez, J.-C. Filiâtre, M. Boespflug
Gr. 2, 7 : G. Schaeffer, P. Chassignet, R. Lebreton
Gr. 3, 8 : J.-P. Tillich, S. Lengrand, C. Cohen
Gr. 4, 9 : B.Werner / J.-M. Steyaert, K. Bhargavan,
B. Tomchuk
Gr. 5, 10 : F. Pottier, Y. Ponty, M. Mezzarobba

Responsable des projets informatiques (PI) :
F. Le Fessant

Aujourd'hui

- I. Organisation du cours.
- II. Généralités
- III. Modularité et Interfaces en Java

Insertion dans le grand tout

	S	O	N	D	J	F	M	A	M	J	J	A	
Année 1	Inc										INF311/321		
Année 2	INF421a INF422		INF421b					INF431 Modex1 Modex2					
Année 3			Master 1					Stage de recherche					
Année 4			Master 2										

I. Organisation du cours

- 18 blocs (9+9), avec amphi de 1h30 et TD ou PC de 2 heures. On trouve tout sur la [Page web du cours](#).
- TD :
 - ▶ programmes à déposer d'une fois sur l'autre (upload), relus par les équipes pédagogiques (2h + 2h → A, B, C, D, E).
 - ▶ Eclipse ; groupes des portables (pour délocalisation).
- Pour les PC : DM à rendre (2h + 2h → A, B, C, D, E).
- **Délégué(e)s** : un(e) ancien(ne) 321, un(e) ancien(ne) 311, un(e) EV2.

Pour mercredi prochain.

Un cours en deux parties

Partie I : (B. Werner) terminer la première phase de l'apprentissage de la programmation et de l'algorithmique.

Sanction : pale CC1 + **projet** (sujets donnés mi-février ; choix (binômes) pour mi-mars ; à rendre à la fin mai ; soutenances entre le 8 et le 18 juin.)

Partie II : (J.-M. Steyaert) réseaux, concurrence, parallélisme, complexité.

Sanction : pale CC2.

Notes finales :

- Note classante finale $CC = (CC_1 + 2CC_2)/3$;
- Note (littérale) de module = $(PI + 2CC)/3 + TD + PC$ avec $TD \in [-1, 1]$, $PC \in [-1, 1]$. **Pas de validation sans projet!**

Emploi du temps prévisionnel (2)

Partie II (réseaux, concurrence, logique, complexité)

12/05 – Bloc 13 : Réseaux II (amphi + PC)

26/05 – Bloc 14 : Réseaux III (amphi + PC)

02/06 – Bloc 15 : Réseaux IV (amphi + TD)

28/05 – remise des PI

09/06 – Bloc 16 : Prog. Dyn. (amphi + PC)

08/06 au 19/06 – soutenance des PI

16/06 – Bloc 17 : Parallélisme (amphi + TD)

23/06 – Bloc 18 : Grappes (amphi + PC révision)

30/06 – pale CC2

Préparation du Futur

Choix des PA et de la 4e année...

Emploi du temps prévisionnel

Partie I (programmation et algorithmique)

27/01 – Bloc 01 : Java I (amphi + TD)

03/02 – Bloc 02 : Java II (amphi + TD)

10/02 – Bloc 03 : Révision complexité (amphi + PC)

17/02 – Bloc 04 : Révision Arbres (amphi + TD)

03/03 – Bloc 05 : Graphes I (amphi + TD)

10/03 – Bloc 06 : Graphes II (amphi + PC)

17/03 – Bloc 07 : Graphes III (amphi + PC + DM) **Choix du PI**

24/03 – Bloc 08 : Graphes IV (amphi + PC)

31/03 – Bloc 09 : Graphes V (amphi + TD + DM)

07/04 – Bloc 10 : Modélisation (amphi + PC révision)

14/04 – pale CC1

Partie II (réseaux, concurrence, logique)

28/04 – Bloc 11 : Preuves (amphi + TD)

05/05 – Bloc 12 : Réseaux I (amphi + TD)

...

Le cours

Achève la formation généraliste en informatique

Contexte :

près du tiers de la croissance mondiale est générée par les technologies de l'information et la communication (TIC)

Taux comparable l'effort de R&D mondial

Même chose pour les débouchés des X!

- Physique : simulations
- Biologie : génomique, tests *in silico* . . .
- Économie : modèles multi-agents, validation expérimentale
- Mécanique : éléments finis, souffleries simulées. . .
- Informatique (!) : méthodes formelles, conception de circuits. . .
- Mathématiques (!!)

Nouveaux résultats, nouvelles approches



Il n'y a pas de rangement plus dense
conjecturé en 1612 (Kepler) . . . prouvé en 1998-2000
(Hales)

Un tout petit morceau de la preuve de Hales

Lemma 751442360

$$2.51^2 \leq x_1 \leq 2.696^2 \rightarrow$$

$$4 \leq x_2 \leq 2.168^2 \rightarrow$$

$$4 \leq x_3 \leq 2.168^2 \rightarrow$$

$$4 \leq x_4 \leq 2.51^2 \rightarrow$$

$$4 \leq x_5 \leq 2.51^2 \rightarrow$$

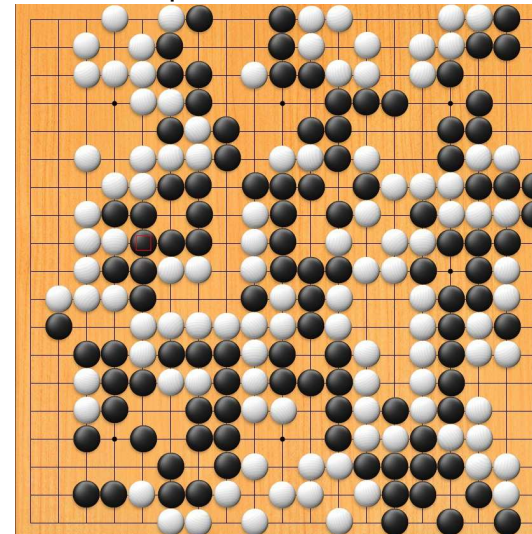
$$4 \leq x_6 \leq 2.51^2 \rightarrow$$

$$\frac{-x_1x_3 - x_2x_4 + x_1x_5 + x_3x_6 - x_5x_6 + x_2(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6)}{\sqrt{4x_2 \left(\begin{array}{l} x_2x_4(-x_2 + x_1 + x_3 - x_4 + x_5 + x_6) + \\ x_1x_5(x_2 - x_1 + x_3 + x_4 - x_5 + x_6) + \\ x_3x_6(x_2 + x_1 - x_3 + x_4 + x_5 - x_6) \\ - x_1x_3x_4 - x_2x_3x_5 - x_2x_1x_6 - x_4x_5x_6 \end{array} \right)}} < \tan\left(\frac{\pi}{2} - 0.74\right)$$

Combinaison Informatique + maths apps. (optimisation. . .)

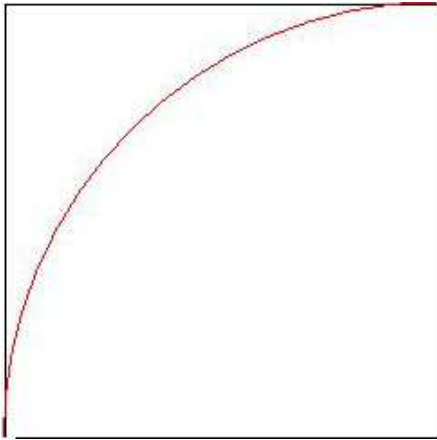
Comment estimer une position au go ?

Encore un peu de culture. . .



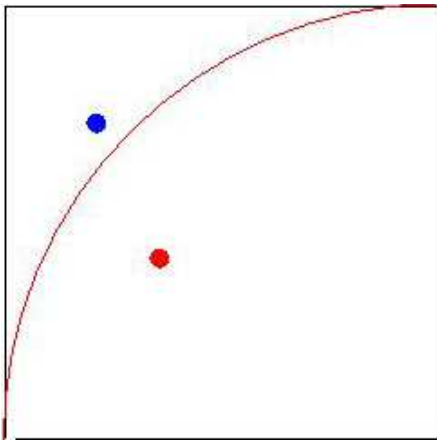
La position favorise-t-elle noir ou blanc ?
difficile pour les ordinateurs
La solution est au casino !

Calculer π au hasard



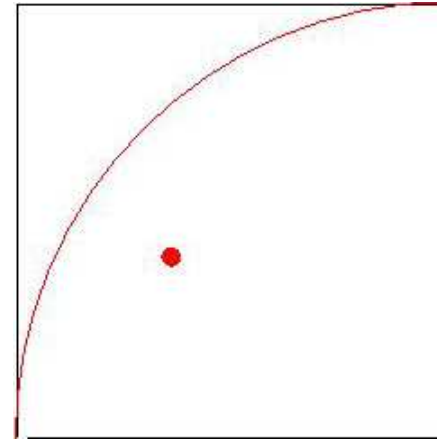
B. Werner – École polytechnique – INF 431 – Promo X2008

Calculer π au hasard



B. Werner – École polytechnique – INF 431 – Promo X2008

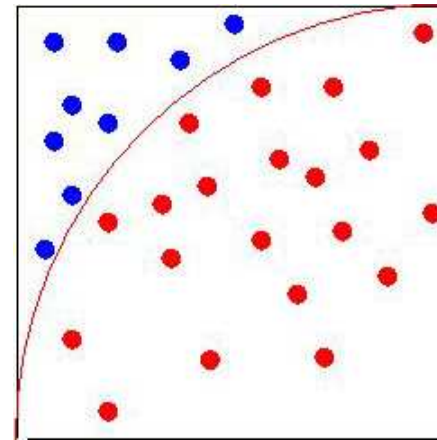
Calculer π au hasard



13/40

B. Werner – École polytechnique – INF 431 – Promo X2008

Calculer π au hasard



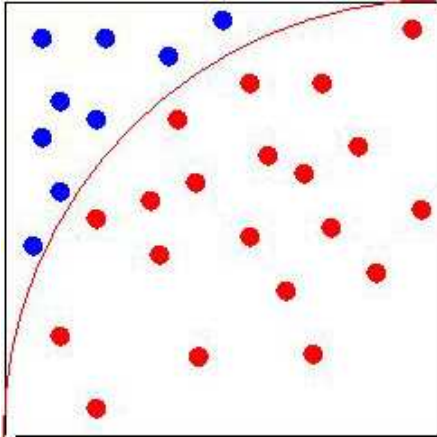
14/40

15/40

B. Werner – École polytechnique – INF 431 – Promo X2008

16/40

Calculer π au hasard



La proportion de rouges tend vers $\pi/4$

En go :

on finit la partie en faisant jouer les deux joueurs **au hasard**

on recommence un grand nombre de fois

on fait des stats

Totalement inhumain, mais efficace

Un bon site de vulgarisation :

<http://interstices.info/>

Des algorithmes partout. . .

Interdisciplinarité :

- de l'ingénierie dans la preuve de Hales
- des maths dans votre appareil photo

Importance de savoir distinguer le faisable de l'infaisable :

culture algorithmique

La question de la taille

La programmation est une des activités les plus complexes jamais entreprises par l'homme.

Windows XP = 50 millions lignes de code,

Linux = 30 millions lignes de code,

⇒ Développement du **Génie Logiciel**

⇒ Nécessité de couper les programmes en morceaux :
modularité

Pourquoi modulariser ?

- Clarifier l'architecture / les concepts
- Partager le travail

Organisation !

des directions *complémentaires* :

- Organiser suivant les tâches
- Organiser suivant les structures de données
- Organiser suivant du code partagé

Modules

Un module a deux parties :

l'interface

- accessible depuis l'extérieur du module,
- e.g. une file a 3 opérations : construction, ajouter, supprimer ;

l'implémentation

- accessible de l'intérieur du module,
- e.g. deux implémentations possibles : tableau circulaire, liste.

Principe d'**encapsulation** : l'extérieur n'a pas à connaître l'implémentation

- on peut changer l'implémentation sans que l'extérieur ne s'en rende compte ;
- on décrit d'abord l'interface, puis chacun travaille à son implantation ;
- simplifie la maintenance : les erreurs sont localisées au module et on ne sature pas l'espace des identificateurs.

Exemples

Un moteur de rendu `html`

La classe `Math` de java

Des bibliothèques graphiques

la base de données de vos contacts téléphoniques

En Java ?

Les classes sont un peu des modules :

- regroupent des fonctions
- composantes privées et publiques

Un exemple, les files :

```
class FIFO{
    ...
    boolean estVide() {...
    void ajouter(int x) {...
    int supprimer() {...
    ...
}
```

Première implantation

```
public class FIFO_T {
    private int debut, fin;
    private boolean pleine, vide;
    private int[] contenu;

    public FIFO_T(int n){
        debut = 0; fin = 0;
        pleine = n == 0; vide = true;
        contenu = new int[n];
    }
    public boolean estVide(){
        return vide;
    }
}
```

```
public void ajouter(int x){
    if(pleine) throw new Error("File Pleine.");
    contenu[fin] = x;
    fin = (fin + 1) % contenu.length;
    vide = false; pleine = fin == debut;
}
```

```
public int supprimer(){
    if(vide)
        throw new Error("File Vide.");
    int res = contenu[debut];
    debut = (debut + 1) % contenu.length;
    vide = fin == debut; pleine = false;
    return res;
}
```

Seconde implantation

```
public class FIFO_L {
    private Liste debut, fin;

    public FIFO_L(){
        debut = null; fin = null;
    }

    public boolean estVide(){
        return (debut == null);
    }

    public void ajouter(int x){
        if(fin == null)
            debut = fin = new Liste(x);
        else{
            fin.suivant = new Liste(x);
            fin = fin.suivant;
        }
    }
}
```

```
public int supprimer(){
    if(debut == null)
        throw new Error("File Vide.");
    else{
        int res = debut.val;
        if(debut == fin)
            debut = fin = null;
        else debut = debut.suivant;
        return res;
    }
}
```

- L'interface est noyée dans la définition
- On doit compter sur l'implémenteur du module pour cacher la partie privée
- On veut pouvoir dire qu'on utilise une certaine définition
FIFO

Il nous faut préciser l'interface du module

Interface pour FIFO :

```
interface FIFO{
    boolean estVide();
    void ajouter(int x);
    int supprimer();
}
```

On décrit la partie publique de la classe utilisée

Comment ça s'utilise ?

Exemple d'utilisation

```
public class C{
    public static void Algo(FIFO f){
        while(! f.estVide()){
            int x = f.supprimer();
            System.out.println(x);
        }
    }
    public static void main(String[] args){
        FIFO_L L = new FIFO_L();
        for(int i = 0; i < 10; i++){
            L.ajouter(i);
        }
        Algo(L);
        FIFO_T T = new FIFO_T(20);
        for(int i = 0; i < 10; i++){
            T.ajouter(i);
        }
        Algo(T);
    }
}
```

```
public class FIFO_L implements FIFO{
    private Liste debut, fin;

    public FIFO_L(){
        ...
    }
}
```

coexiste avec

```
public class FIFO_T implements FIFO{
    private int[] contenu;
    ...
}
```

Spécialisations multiples

```
interface Pile{
    boolean estVide();
    void empiler(int x);
    void depiler();
}
```

On peut réaliser une classe de liste qui implante pile et file (~LinkedList):

```
public class MaListe implements FIFO, Pile{
    ...
}
```

- Un peu de l'héritage multiple
- Attention à l'espace des noms

Points d'interfaces

- Une **interface** est une classe abstraite qui n'a pas de champs sauf des constantes (**static final**). Ses méthodes sont abstraites et publiques.
- Une interface peut **spécialiser** une ou plusieurs interfaces (avec **extends**).
- Une classe peut **implémenter** une ou plusieurs interfaces (avec **implements**).
- Notion différente des interfaces de Modula, ML, Ada, Mesa. En Java, la classe qui les implémente porte un nom différent.
- **Pas** de fonctions statiques, constructeur ou champs de données modifiables dans une interface.
- Les interfaces sont une bonne manière d'imposer des fonctionnalités dans une classe.

Paramétrer nos modules

Si on veut des files de String? de List?

On ne veut pas tout refaire

On crée une **classe générique**

```
public class FIFO_LL <T> {
    private LinkedList<T> contenu;
    ...
}
```

Comment bien spécifier une classe générique ?

avec une interface générique :

```
interface FIFO <T> {
    boolean estVide();
    void ajouter(T x);
    T supprimer();
}
```

```
public class FIFO_LL <T> implements FIFO<T> {
    private LinkedList<T> contenu;
    ...
}
```

...

Restreindre à certaines classes :

Comment empêcher l'application à une classe trop générale ?

Il suffit de donner une **borne supérieure** à la classe utilisable, comme dans

```
public class Box<T extends Number> {...
```

On ne pourra alors utiliser la boîte que pour des sous-classes de la classe `Number`.

Pour résumer

- Nécessité de structurer le code
- En java, les *classes* restent l'outil premier
- Les *interfaces* permettent de désigner des ensembles de classes
- possibilité de paramétrer par des *génériques*

Plein de bibliothèques

Force de java : de nombreuses bibliothèques

Par exemple :

L'interface `Collection<T>` est implémentée par les classes `AbstractList<T>`, `HashSet<T>`, `ArrayList`, `LinkedList`...

spécialisée par les interfaces `List<T>`, `Set<T>`...

[On passe pas mal de temps dans la doc \(à cliquer\)](#)

Quand on a compris les principes, on peut effectivement réutiliser beaucoup de code.

Merci à François Morain pour la réutilisation de code pour les transparents.

La suite

Cet après-midi : interfacier en TD

La semaine prochaine : les objets au fond des yeux

Rappel : noms des délégué(e)s pour le début du prochain ampli. Un(e) délégué(e) par groupe.