

Initiation à la programmation Java - IP2

Rappels de cours No 1 et 2

Yan Jurski

4 février 2020



Compétences acquises

- Vous savez maintenant définir et construire des objets qui agrègent des données plus simples
- Vous comprenez que leur utilisation permet d'améliorer la qualité d'une modélisation, en introduisant du relief.
- Vous maîtrisez les syntaxes liées au mot clé **static** :
 - A.f(), A.x, a.y, a.g()
- Vous comprenez qu'on distingue des portions de code :
 - correspondant à un service (**public**),
 - ou masquées (**private**), qui restent internes ("brouillon", protection)

Compétences acquises

- Vous savez définir et construire des objets qui agrègent des données plus simples

fichier : A.java

```
public class A{  
    String info;  
    int [] data;  
    boolean ok;  
    A(int x){  
        info="Un objet A",  
        data = new int[x];  
        ok=true;  
    }  
}  
// et ailleurs, une construction :  
A a= new A(100);
```

- Vous comprenez que leur utilisation permet d'améliorer la qualité d'une modélisation.

Compétences acquises

- Vous savez définir et construire des objets qui agrègent des données plus simples

fichier : A.java

```
public class A{  
    String info;  
    int [] data;  
    boolean ok;  
    A(int x){  
        info="Un objet A",  
        data = new int[x];  
        ok=true;  
    }  
    // et ailleurs, une construction :  
    A a= new A(100);
```

- Vous comprenez que leur utilisation permet d'améliorer la qualité d'une modélisation. . . A, B, Point, Triangle, Cercle, Automobile, . . .

Compétences acquises

- Vous savez définir et construire des objets qui agrègent des données plus simples
- Vous comprenez que leur utilisation permet d'améliorer la qualité d'une modélisation. ... A, B, Point, Triangle, Cercle, Automobile, ...
- Vous maîtrisez les syntaxes liées au mot clé **static** :

A.f(), A.x, a.y, a.g()

- A.f() : la méthode f est déclarée statique dans la classe A
 - A.x : l'attribut x est déclaré statique dans la classe A
 - a.y : les objets de la classe de a disposent d'un attribut non statique y
 - a.g() : la méthode g s'exécute dans le contexte de l'objet a qui s'identifie localement (si besoin) par **this**
-
- Vous comprenez qu'on distingue des parties :
 - correspondant à un service (**public**),
 - ou masquées (**private**), qui restent internes ("brouillon", protection)

LES BONNES MANIERES



- Un objet peut être utilisé de deux façons :
 - comme simple argument, il est "neutre"
 - comme sujet, acteur responsable
- Vous devez penser à distinguer :
 - ce qui est propre à l'objet (données, comportement),
 - et ce qui est partagé, global (**static**)
- Respectez sa "vie privée"
- Définir le contrat d'interface (**public**), le reste est **private**
Les difficultés sont surmontées par des choix internes
(votre travail), invisibles à l'utilisateur externe
- Vous pourrez livrer une librairie documentée, intelligible

Initiation à la programmation Java

IP2 - Cours No 3

Yan Jurski

4 février 2020

Exercices Modélisations

(1) - Partiel 2018

- Un binôme est un objet qui se caractérise par une paire de String
- Ils sont numérotés de façon unique et automatique dès leur création
- Leur représentation interne est normalisée : chaque paire a toujours sa première chaîne plus petite ou égale dans l'ordre lexicographique que sa seconde chaîne
- La constitution de ces binômes pourra changer avec le temps.
- On souhaite conserver un représentant des plus petits binômes (au sens lexicographique) qui ont pu être créés dans le passé

Ecrire une classe Binôme en justifiant vos choix.

Ecrivez aussi des méthodes auxiliaires utiles à leur manipulation.

Exercices Modélisations

(1) - Partiel 2018

- Un binôme est un objet qui se caractérise par une paire de String

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    public Binome (String a,String b){  
        s1=a; s2=b;  
    }  
}
```

Exercices Modélisations

(1) - Partiel 2018

- Ils sont numérotés de façon unique et automatique à leur création

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    public Binome (String a,String b){  
        s1=a; s2=b;  
        this.num=Binome.nextNum++;  
    }  
}
```

Exercices Modélisations

(1) - Partiel 2018

- **Leur représentation est normalisée** : chaque paire a toujours sa première chaîne plus petite ou égale dans l'ordre lexicographique que sa seconde chaîne

dans Fichier : Binome.java

```
// la chaîne a est supérieure stricte à b lexicographiquement ?  
private static boolean superieur(String a, String b){  
    int la= a.length(), lb=b.length();  
    int min_l=Math.min(la,lb);  
    for (int i=0;i<min_l;i++)  
        if (a.charAt(i) > b.charAt(i)) return true;  
        else if (a.charAt(i) < b.charAt(i) ) return false;  
    return (la > lb);  
}
```

Exercices Modélisations

(1) - Partiel 2018

- **Leur représentation est normalisée** : chaque paire a toujours sa première chaîne plus petite ou égale dans l'ordre lexicographique que sa seconde chaîne

dans Fichier : Binome.java

```
// a supérieur strict à b lexicographiquement
private static boolean superieur(String a, String b){
    ...
}

private void normalise(){
    if (Binome.superieur(this.s1, this.s2)) {
        String tmp=this.s1;
        this.s1=this.s2;
        this.s2=tmp;
    }
}
```

Exercices Modélisations

(1) - Partiel 2018

- **Leur représentation est normalisée** : chaque paire a toujours sa première chaîne plus petite ou égale dans l'ordre lexicographique que sa seconde chaîne

dans Fichier : Binome.java

```
public Binome (String a,String b){  
    s1=a;  
    s2=b;  
    this.normalise();  
    num=nextNum++;  
}  
private static boolean superieur(String a, String b){...}  
  
private void normalise(){...}
```

Exercices Modélisations

(1) - Partiel 2018

- La constitution de ces binômes pourra changer avec le temps

dans Fichier : Binome.java

```
public void change1(String x){  
    this.s1=x;  
    this.normalise();  
}  
  
public void change2(String x){  
    s2=x;  
    normalise();  
}
```

Exercices Modélisations

(1) - Partiel 2018

- On souhaite conserver un représentant des plus petits binômes (au sens lexicographique) qui ont pu être créés dans le passé

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    private static Binome plusPetitEver=null;  
    private void normalise(){  
        if (Binome.superieur(this.s1,this.s2)) {  
            String tmp=this.s1;  
            this.s1=this.s2;  
            this.s2=tmp;  
        }  
        conservePlusPetit();  
    }  
    ...  
}
```

Exercices Modélisations

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    private static Binome plusPetitEver=null;  
    private void normalise(){...}  
    private static boolean superieur(String a, String b){...}  
    private void conservePlusPetit(){  
        if ( (plusPetitEver==null) || estLePlusPetit())  
            plusPetitEver=this;  
    }  
    // compare le binôme courant avec le plus petit jamais créé  
    private boolean estLePlusPetit(){ // lexicographiquement  
        if (Binome.superieur(s1,plusPetitEver.s1)) return false;  
        if (! s1.equals(plusPetitEver.s1)) return true;  
        if (Binome.superieur(s2,plusPetitEver.s2)) return false;  
        if (! s2.equals(plusPetitEver.s2)) return true;  
        return false;  
    }  
}
```

Exercices Modélisations

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    private static Binome plusPetitEver=null;  
    private void normalise(){...}  
    private static boolean superieur(String a, String b){...}  
    private void conservePlusPetit(){  
        if ( (plusPetitEver==null) || estLePlusPetit())  
            plusPetitEver=this; // malheureusement mauvaise réalisation  
    }  
    private boolean estLePlusPetit(){ ... }  
}
```

```
Binome b= new Binome ("aa","zz");  
// il est supposé seul, il est donc aussi le plusPetitEver  
b.change1("bb");  
// b ET plusPetitEver ont tous les deux changés !
```

Exercices Modélisations

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    private static Binome plusPetitEver=null;  
    private void normalise(){...}  
    private static boolean superieur(String a, String b){...}  
    private void conservePlusPetit(){  
        if ( (plusPetitEver==null) || estLePlusPetit())  
            plusPetitEver=new Binome(s1,s2); // avec une copie c'est mieux ?  
    }  
    private boolean estLePlusPetit(){ ... }  
}
```

```
Binome b= new Binome ("aa","zz");  
// ... ne termine pas ... le constructeur appelle normalise, qui  
    appelle conservePlusPetit qui appelle un constructeur ...  
// Exception in thread "main" java.lang.StackOverflowError
```

Exercices Modélisations

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    private static Binome plusPetitEver=null;  
    private void normalise(){...}  
    private static boolean superieur(String a, String b){...}  
    private void conservePlusPetit(){  
        if ( (plusPetitEver==null) || estLePlusPetit())  
            plusPetitEver=new Binome(s1,s2); // avec une copie c'est mieux ?  
    }  
    private boolean estLePlusPetit(){ ... }  
}
```

```
Binome b= new Binome ("aa","zz");  
// ... ne termine pas ... le constructeur appelle normalise, qui  
    appelle conservePlusPetit qui appelle un constructeur ...  
// Exception in thread "main" java.lang.StackOverflowError
```

Il faut faire une copie différemment, avec un **autre constructeur**

Exercices Modélisations

Fichier : Binome.java

```
public class Binome {  
    private String s1,s2;  
    private final int num;  
    private static int nextNum=0;  
    private static Binome plusPetitEver=null;  
    ...  
    private Binome( Binome b) {s1=b.s1; s2=b.s2; num=b.num;}  
    ...  
    private void conservePlusPetit(){  
        if ( (plusPetitEver==null) || estLePlusPetit())  
            plusPetitEver=new Binome(this); // un constructeur dédié  
    }  
    private boolean estLePlusPetit(){ ... }  
}
```

```
Binome b= new Binome ("aa","zz");  
b.change1("bb"); // n'impacte pas plusPetitEver  
// c'est le mieux qu'on puisse faire étant données les spécifications
```

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



- tous les tickets ont toujours le même prix
 - un numéro de série distingue les tickets
 - deux cases à gratter masquent des entiers "Gain", "Nul si découvert" (qui est un nombre aléatoire à 3 chiffres)
 - On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
 - On ne peut pas les masquer une fois révélées
 - Chaque valeur lisible est définitive, et déterminée dès le départ
- Ecrire la classe **Banco** (attributs, constructeurs, méthodes ...)

Probabilités de gains :

- à 75 % perdant
- à 24 % de 2 €
- à 1 % de 1000 €

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



Probabilités de gains :

- à 75 % perdant
- à 24 % de 2 €
- à 1 % de 1000 €

- tous les tickets ont toujours le même prix
- un numéro de série distingue les tickets
- deux cases à gratter masquent des entiers "Gain", "Nul si découvert" (qui est un nombre aléatoire à 3 chiffres)
- On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
- On ne peut pas les masquer une fois révélées
- Chaque valeur lisible est définitive, et déterminée dès le départ

```
public class Banco {  
    public static final int prix=1;  
    ...  
}
```

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



Probabilités de gains :

- à 75 % perdant
- à 24 % de 2 €
- à 1 % de 1000 €

- un numéro de série distingue les tickets
- deux cases à gratter masquent des entiers "Gain", "Nul si découvert" (qui est un nombre aléatoire à 3 chiffres)
- On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
- On ne peut pas les masquer une fois révélées
- Chaque valeur lisible est définitive, et déterminée dès le départ

```
public class Banco {  
    public static final int prix=1;  
    private static int nb=0;  
    public final int myNumber;  
    ...  
    public Banco(){  
        myNumber=nb++;  
        ...  
    }  
}
```

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



Probabilités de gains :

- à 75 % perdant
- à 24 % de 2 €
- à 1 % de 1000 €

- deux cases à gratter masquent des entiers "Gain", "Nul si découvert" (qui est un nombre aléatoire à 3 chiffres)
- On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
- On ne peut pas les masquer une fois révélées
- **Chaque valeur lisible est définitive, et déterminée dès le départ**

```
public class Banco {  
    public static final int prix=1;  
    private static int nb=0;  
    public final int myNumber;  
    final int gain;  
    final int nullSiDécouvert;  
    boolean gainGratté, nullGratté;  
    public Banco(){  
        myNumber=nb++;  
        gainGratté=false; nullGratté=false;  
        gain=... // à faire  
        nullSiDecouvert=... // à faire  
    }  
}
```

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



Probabilités de gains :

- à 75 % perdant
- à 24 % de 2 €
- à 1 % de 1000 €

- deux cases à gratter masquent des entiers "Gain", "Nul si découvert" (**qui est un nombre aléatoire à 3 chiffres**)
- On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
- On ne peut pas les masquer une fois révélées

```
public class Banco {  
    public static final int prix=1;  
    private static int nb=0;  
    public final int myNumber;  
    final int gain;  
    final int nullSiDécouvert;  
    boolean gainGratté, nullGratté;  
    public Banco(){  
        myNumber=nb++;  
        gainGratté=false; nullGratté=false;  
        gain=... // à faire  
        nullSiDécouvert=(int) (Math.random()*1000);  
    }  
    ...  
}
```

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



- On ne peut pas voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
- On ne peut pas les masquer une fois révélées

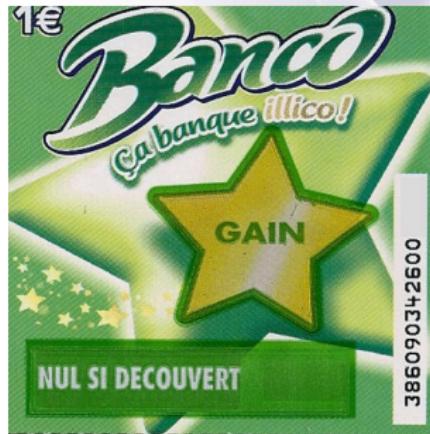
Probabilités de gains :

- à 75 % perdant
- à 24 % de 2 €
- à 1 % de 1000 €

```
public class Banco {  
    public static final int prix=1;  
    private static int nb=0;  
    public final int myNumber;  
    final int gain;  
    final int nullSiDécouvert;  
    boolean gainGratté, nullGratté;  
    public Banco(){  
        myNumber=nb++;  
        gainGratté=false; nullGratté=false;  
        double p=Math.random(); // dans [0;1[  
        if (p<=.75) gain=0; else if (p<=.99) gain = 2; else gain = 1000;  
        nullSiDécouvert=(int) (Math.random()*1000);  
    }  
    ...  
}
```

Exercices Modélisations

(2) - Examen 2018 - Banco - Jeu de grattage



- **On ne peut pas** voir au travers de ces cases tant qu'elles n'ont pas été révélées (sinon retourne -1)
- **On ne peut pas** les masquer une fois révélées

```
public class Banco {  
    public static final int prix=1;  
    private static int nb=0;  
    public final int myNumber;  
    private final int gain;  
    private final int nullSiDécouvert;  
    private boolean gainGratté, nullGratté;  
    public Banco(){  
        myNumber=nb++;  
        double p=Math.random();  
        if (p<=.75) gain=0; else if (p<=.99) gain = 2; else gain = 1000;  
        nullSiDécouvert=(int) (Math.random()*1000);  
        gainGratté=false; nullGratté=false;  
    }  
    public int getGain(){  
        if (gainGratté) return gain; else return -1;  
    }  
    public int getNulSi(){  
        if (nullGratté) return nullSiDécouvert; else return -1;  
    }  
    public void gratteGain(){ gainGratté=true;}  
    public void gratteNulSi(){ nullGratté=true;}  
}
```

Exercices Modélisations

(3) - Session 2 - 2018 - Compteur kilométrique



- Une entreprise *Lambda* les fabrique
- Sur les compteurs est imprimée une unité : soit km/h, soit miles/h
- ils peuvent recevoir un signal qui leur indique la vitesse réelle
- Un bouton remet à zéro le petit compteur
- ils reçoivent pour information le temps passé à la vitesse actuelle
- tous les 10000 km afficher "révision à faire".
- si l'entreprise se fait racheter, tous les compteurs changent le nom de leur marque
- Ecrire la classe **Compteur** (attributs, constructeurs, méthodes ...)

Exercices Modélisations

(3) - Session 2 - 2018 - Compteur kilométrique



- Une entreprise *Lambda* les fabrique
- Sur les compteurs est imprimée une unité : soit km/h, soit miles/h
- ils peuvent recevoir un signal qui leur indique la vitesse réelle
- Un bouton remet à zéro le petit compteur
- ils reçoivent pour information le temps passé à la vitesse actuelle
- tous les 10000 km afficher "révision à faire"
- si l'entreprise se fait racheter, tous les compteurs changent le nom de leur marqueur

```
public class Compteur {  
    private static String entreprise="Lambda";  
    ...  
    public static void changeCompany(String s){  
        Compteur.entreprise=s;  
    }  
}
```

Exercices Modélisations

(3) - Session 2 - 2018 - Compteur kilométrique



- Sur les compteurs est **imprimée** une unité : soit km/h, soit miles/h
- ils peuvent recevoir un signal qui leur indique la vitesse réelle
- Un bouton remet à zéro le petit compteur
- ils reçoivent pour information le temps passé à la vitesse actuelle
- tous les 10000 km afficher "révision à faire"

```
public class Compteur {  
    private static String entreprise="Lambda";  
    public final String unité;  
  
    public Compteur(boolean kmH){ // une façon de déterminer l'unité  
        if (kmH) unité="Km/h"; else unité="Miles/h";  
    }  
    public static void changeCompany(String s){  
        Compteur.entreprise=s;  
    }  
}
```

Exercices Modélisations

(3) - Session 2 - 2018 - Compteur kilométrique



- ils peuvent recevoir un signal qui leur indique la **vitesse** réelle
- Un bouton remet à zéro le **petit compteur**
- ils reçoivent pour information le temps passé à la vitesse actuelle
- **tous les** 10000 km afficher "révision à faire".

On trouve, dans l'énoncé, les notions qui indiquent les données à stocker

```
public class Compteur {  
    private static String entreprise="Lambda";  
    public final String unité;  
    int vitesse_courante=0;  
    int compteur_inter=0, compteur_total=0;  
    // rq : on peut faire cette initialisation lors de la déclaration  
    public Compteur(boolean kmH){  
        if (kmH) unité="Km/h"; else unité="Miles/h";  
    }  
    public static void changeCompany(String s){  
        Compteur.entreprise=s;  
    }  
}
```

Exercices Modélisations

(3) - Session 2 - 2018 - Compteur kilométrique



- ils **peuvent recevoir un signal** qui leur indique la vitesse réelle
- **Un bouton** remet à zéro le petit compteur
- ils reçoivent pour information le temps passé à la vitesse actuelle
- tous les 10000 km afficher "révision à faire".

```
public class Compteur {  
    private static String entreprise="Lambda";  
    public final String unité;  
    private int vitesse_courante=0;  
    private int compteur_inter=0, compteur_total=0;  
  
    public Compteur(boolean kmH){  
        if (kmH) unité="Km/h"; else unité="Miles/h";  
    }  
  
    public void set_speed(int v){ vitesse_courante=v; }  
    public void reset(){ compteur_inter=0; }  
    public static void changeCompany(String s){  
        Compteur.entreprise=s;  
    }  
}
```

Exercices Modélisations

(3) - Session 2 - 2018 - Compteur kilométrique



- ils reçoivent pour information le temps passé à la vitesse actuelle
- tous les 10000 km afficher "révision à faire".

```
public class Compteur {  
    private static String entreprise="Lambda";  
    public final String unité;  
    private int vitesse_courante=0;  
    private int compteur_inter=0, compteur_total=0;  
  
    public Compteur(boolean kmH){  
        if (kmH) unité="Km/h"; else unité="Miles/h";  
    }  
    public void set_speed(int v){ vitesse_courante=v; }  
    public void reset(){ compteur_inter=0; }  
    public void time_elapsed(double t){  
        int delta=(int)(vitesse_courante*t);  
        compteur_inter+=delta;  
        int old_compteur=compteur_total;  
        compteur_total+=delta;  
        if ((compteur_total/10000) != (old_compteur/10000) )  
            System.out.println("Il faut réviser :");  
    }  
    public static void changeCompany(String s){  
        Compteur.entreprise=s;  
    }  
}
```

Exercice (difficile) - Relation clé/serrure



- Une serrure est construite pour fonctionner avec une toute nouvelle clé (qui n'a jamais été utilisée)
- On peut demander une unique fois à une serrure de nous révéler sa clé
- On ne peut pas forger une clé à partir d'une serrure.
- Si on a oublié d'où elle provient, une clé seule ne peut pas indiquer avec quelle serrure elle fonctionne. Mais une serrure peut tester si oui ou non une clé convient.
- On veut pouvoir dupliquer une clé.

Exercice (difficile) - Relation clé/serrure



- Une serrure est construite pour fonctionner avec une toute nouvelle clé (qui n'a jamais été utilisée)
- On peut demander **une unique fois** à une serrure de nous révéler sa clé
- On ne peut pas forger une clé à partir d'une serrure.
- Si on a oublié d'où elle provient, **une clé seule ne peut pas indiquer avec quelle serrure elle fonctionne**. Mais une serrure peut tester si oui ou non une clé convient.
- On veut pouvoir dupliquer une clé.

Exercice - Relation clé/serrure



Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c; // master key  
    // reste à la récupérer UNE fois  
}
```

Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    // pas d'accesseur !  
}
```

Exercice - Relation clé/serrure



Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c; // master key  
    private boolean cléDisponible ;  
    public Clé getClé(){ // accesseur unique  
        if (cléDisponible) {  
            cléDisponible=false;  
            return c;  
        } else return null;  
    }  
}
```

Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
}
```

Exercice - Relation clé/serrure



- Une serrure est construite pour fonctionner avec **une toute nouvelle clé** (qui n'a jamais été utilisée)
- On peut demander une unique fois à une serrure de nous révéler sa clé
- Sinon, on ne peut pas forger une clé à partir d'une serrure.
- Si on a oublié d'où elle provient, une clé seule ne peut pas indiquer avec quelle serrure elle fonctionne. Mais une serrure peut tester si oui ou non une clé convient.
- On veut pouvoir dupliquer une clé.

Exercice - Relation clé/serrure



Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c;  
    private boolean cléDisponible;  
    public Serrure(){  
        cléDisponible=true;  
        c=new Clé(this); // encore jamais utilisée  
        // donc constructeur public  
    }  
    public Clé getClé(){...}  
}
```

Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    public Clé(Serrure s){ // nécessaire  
        this.couplée=s; // mais encore trop permissif  
    }  
}
```

Exercice - Relation clé/serrure



Le constructeur public Clé(Serrure s)

- est nécessaire, car appelé dans le constructeur de Serrure
- est, en l'état, trop permissif car il nous permet de forger une clé à partir de toutes serrures.
- on ne peut pas empêcher l'appel au constructeur, mais on peut neutraliser le couplage. La clé construite n'ouvrira donc aucune serrure si on considère que la construction est illégale

Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c;  
    private boolean cléDisponible;  
    public Serrure(){  
        cléDisponible=true;  
        c=new Clé(this); // encore jamais utilisée  
    }  
    public boolean estSansClé(){  
        if (this.c==null) return true;  
        else return false;  
    }  
    public Clé getClé(){...}  
}
```



Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    public Clé(Serrure s){  
        if (s.estSansClé()) this.couplée=s;  
        else this.couplée=null; // clé inutilisable  
    }  
}
```

Exercice - Relation clé/serrure



- Une serrure est construite pour fonctionner avec une toute nouvelle clé (qui n'a jamais été utilisée)
- On peut demander une unique fois à une serrure de nous révéler sa clé
- Sinon, on ne peut pas forger une clé à partir d'une serrure.
- Si on a oublié d'où elle provient, une clé seule ne peut pas indiquer avec quelle serrure elle fonctionne. Mais une serrure peut tester si **oui ou non une clé convient**.
- On veut pouvoir **duplicer** une clé.

Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c;  
    private boolean cléDisponible;  
    public Serrure(){  
        cléDisponible=true;  
        c=new Clé(this); // encore jamais utilisée  
    }  
    public boolean estSansClé(){...}  
    public Clé getClé(){...}  
}
```



Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    public Clé(Serrure s){  
        if (s.estSansClé()) this.couplée=s;  
        else this.couplée=null; // clé inutilisable  
    }  
    public Clé duplicate(){  
        return new Clé(couplée); // ne convient pas  
    } // nécessite d'un autre constructeur  
} // de signature différente
```

Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c;  
    private boolean cléDisponible;  
    public Serrure(){...}  
    public boolean estSansClé(){...}  
    public Clé getClé(){...}  
}
```



Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    public Clé(Serrure s){  
        if (s.estSansClé()) this.couplée=s;  
        else this.couplée=null; // clé inutilisable  
    }  
    private Clé(Clé x){  
        this.couplée=x.couplée;  
    }  
    public Clé duplicate(){  
        return new Clé(this);  
    }  
}
```

Exercice - Relation clé/serrure



- Une serrure est construite pour fonctionner avec une toute nouvelle clé (qui n'a jamais été utilisée)
- On peut demander une unique fois à une serrure de nous révéler sa clé
- Sinon, on ne peut pas forger une clé à partir d'une serrure.
- Si on a oublié d'où elle provient, une clé seule ne peut pas indiquer avec quelle serrure elle fonctionne. Mais une serrure peut tester si **oui ou non une clé convient**.
- On veut pouvoir dupliquer une clé.

Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c;  
    ...  
    public boolean convient(Clé x){  
        if (c==x) {  
            System.out.println("Master Key");  
            return true;  
        } // et le cas des copies ?  
        if (x.couplée==this) // acces interdit !  
            return true;  
    }  
}
```



Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    ...  
}
```

Fichier : Serrure.java

```
public class Serrure {  
    private final Clé c;  
    ...  
    public boolean convient(Clé x){  
        if (c==x) {  
            System.out.println("Master Key");  
            return true;  
        }  
        if (x==null) return false;  
        return x.convient(this);  
    }  
}
```



Fichier : Clé.java

```
public class Clé {  
    private final Serrure couplée;  
    ...  
    public boolean convient(Serrure s){  
        return this.couplée==s;  
    }  
}
```

Rq : méthodes pas forcément de même noms



Exercice - Relation clé/serrure

Test de l'ensemble



Fichier : Test.java

```
public class Test {  
    public static void main(String [] args){  
        Serrure lock=new Serrure();  
        Clé myKey=lock.getClé(); // ok  
        Clé cléBidon=lock.getClé(); // null  
        Clé cléBidon2=new Clé(lock); // sans serrure  
        Serrure s=myKey.couplée(); // interdit  
        Serrure s2=myKey.getSerrure(); // n'existe pas  
        System.out.println(lock.convient(myKey));  
        // true Master Key  
        Clé monPass=myKey.duplicate(); // ok  
        System.out.println(lock.convient(monPass));  
        // true  
        System.out.println(lock.convient(cléBidon));  
        System.out.println(lock.convient(cléBidon2));  
        // false, false  
    }  
}
```

Prochain cours

- D'autres liaisons complexes entre objets
- le cas important des listes chaînées