

Correction TD n°1

Java Avancé

—M1 Apprentissage—

Révisions




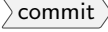

Objets, références, égalité, encapsulation, mutabilité, redéfinition etc.

► Exercice 1. Git

Afin :

- d’avoir votre travail toujours disponible (chez vous, à la fac...),
- de travailler à plusieurs sur du même code (projet),
- garder un historique des différentes versions de votre code source,
- s’habituer à des outils,

vous allez créer un compte GIT sur GitHub (sauf si vous avez déjà un compte). Ils offrent la possibilité d’avoir des dépôts **privés** (important) et **gratuits**.

1. Se rendre à l’adresse https://classroom.github.com/a/_m3NL1eh trouvez votre email pour linker votre compte github à l’assignement. Cela crée un dépôt **privé** type `td-java-xxxx`, où `xxx` est votre login.
2. Ouvrez un terminal, créez un dossier `java`, et faites les quelques instructions en ligne de commande données par la page afin d’initialiser votre dépôt sur votre compte (`init`, `add`, `commit`, `remote`, `push`).
3. Créer un projet `java` dans le répertoire où vous avez créé votre dépôt GIT. Eclipse devrait détecter automatiquement qu’il s’agit d’un dépôt GIT.
4. Créer une classe dans le projet `Java`. Écrire une méthode `main` vide.
5. Dire à GIT que vous voulez l’ajouter à la gestion de version avec clic droit,  . Ceci ajoute uniquement le fichier, les autres fichiers seront ignorés par GIT. Il est possible d’ajouter un dossier à l’index...
6. Envoyez vos changements sur le serveur avec clic droit,    (avec un message (on utilise des messages en anglais généralement, restez cohérents)). Vérifier sur l’interface web de GitHub de votre projet (dans source) que le fichier s’y trouve bien.
7. Ajouter un `print` dans la méthode `main`, sauvegardez le fichier, puis dans le terminal, dans le dossier où se situe votre méthode `main`, faire `git diff`. Qu’observez-vous ?
8. Dans un terminal, faites `cd /tmp` pour vous rendre dans le répertoire temporaire, puis cloner le dépôt que vous venez de créer avec la commande `git clone ADRESSE DE VOTRE DEPOT`

9. Faites une modification **au début** d'un des fichiers de votre dépôt **dans le répertoire tmp** et commitez avec `git commit . -m ''debut''`, puis envoyez avec `git push`.
10. Faites une modification **à la fin** du même fichier mais dans le **répertoire original** de votre dépôt, puis tentez d'envoyer. Que se passe t-il ?
11. Toujours dans le même répertoire original, faites maintenant `git pull`. Que contient le fichier modifié ? Que se serait-il passé si vous aviez modifié sur la même ligne ? Plus d'info ici <https://help.github.com/articles/resolving-a-merge-conflict-using-the-command-line/>.
12. Supprimez le répertoire dans `/tmp` pour éviter tout soucis.

► Exercice 2. Eclipse

1. Que se passe lorsque vous tapez `main` puis `[ctrl]` et barre d'espace dans une classe vide ?
2. Même question en tapant `toStr` puis `[ctrl]` et espace dans une classe ?
3. Que se passe t-il si l'on tape `sysout` et que l'on appuie sur `[ctrl]` et espace dans une méthode `main` ?
4. Créer un champ `toto` de type `int` dans une classe. Que se passe t-il si l'on tape `get` puis `[ctrl]` et espace dans la classe ? Et `set` puis `[ctrl]` et espace ?
5. Sélectionner le nom de la classe. Que se passe t-il si l'on tape `[⌘]` (alt)+`[↑]`+R ? Meme question avec le champ `toto`.
6. Il peut être utile de voir le code source utilisé par la JDK. Pour cela, télécharger le fichier `src.zip` sur le site d'oracle (sur les machines du CRIO c'est déjà dans `/usr/local/jdk***/src.zip`) et attacher-le dans `Window` » `Preferences` » `Installed JREs` » `Edit` » `rt.jar` » `Source Attachment`. Déclarer une variable de type `String` et cliquer sur `String` en maintenant la touche `[ctrl]`. Que se passe-t-il ?

► Exercice 3. Recherche de nombres premiers

Cet exercice constitue un prérequis pour le cours de Java Avancé, si vous ne parvenez pas à le faire, manifestez-vous rapidement.

Rappel : pour tester si un nombre p est premier, une méthode simple consiste à tester tous ses diviseurs potentiels entre 2 et \sqrt{p} . Le nombre p est premier si et seulement s'il n'existe aucun $x \in \{2 \dots \sqrt{p}\}$ tel que le *reste* de la division $\frac{p}{x}$ est nul.

1. Créer une classe qui s'appelle `PrimeCollection` et a en champ privé une instance de `java.util.ArrayList<Integer>`.
2. Créer une nouvelle méthode `initRandom(n, m)` pour initialiser une `PrimeCollection` avec `n` entiers tirés aléatoirement entre 1 et `m`. (Pour générer des nombres aléatoires, utiliser `java.util.Random`).

3. Ajouter une méthode privée `isPrime(p)` qui retourne `true` si l'entier `p` passé en paramètre est premier ou `false` sinon.
4. Écrire une méthode `printPrimes` qui affiche uniquement les nombres premiers de la collection.
5. Écrire une méthode `main` pour qui génère une collection de 100 entiers aléatoires tirés entre 1 et 100, et affiche ceux qui sont premiers grâce aux méthodes précédentes.



```

1 import java.util.ArrayList;
2 import java.util.Random;
3
4 public class PrimeCollection extends ArrayList<Integer>{
5     private static final long serialVersionUID = 8920744473480450385L;
6
7     private static boolean isPrime(int p) {
8         for(int x=2 ; x*x<=p ; ++x) {
9             if (p % x == 0) return false;
10        }
11        return true;
12    }
13
14    public void initRandom(int n, int m) {
15        Random r = new Random();
16        for (int i=0;i<n;++i) {
17            add(r.nextInt(m));
18        }
19    }
20
21
22    public static void main(String[] args) throws InterruptedException {
23        PrimeCollection pc = new PrimeCollection();
24        pc.initRandom(10, 20);
25        for each
26        if
27    }
28 }
29

```



► Exercice 4. Point

Le but est d'écrire une classe représentant des coordonnées cartésiennes.

1. Créer une classe `Point` avec deux champs **privés** `x` et `y`. Écrire une méthode `main` avec le code suivant :

```

1 Point p=new Point();
2 System.out.println(p.x+" "+p.y)

```

Pourquoi cela fonctionne t-il ?

2. Créer une classe `TestPoint` avec un `main` ayant le même code que précédemment. Que se passe-t-il ? Comment peut-on y remédier ?

3. Pourquoi il faut toujours que les champs d'une classe soient privés ?
4. Qu'est-ce qu'un accesseur ? Doit-on le faire ici ?
5. Créer un constructeur prenant les coordonnées du point en paramètre (appelés `px` et `py`). Quel est le problème au niveau du `main` ?
6. Modifier les paramètres du constructeur pour les appeler `x` et `y`. Que se passe-t-il ?
7. On veut pouvoir connaître à tout moment le nombre de points qui ont été créés. Comment faire ?
8. Écrire un autre constructeur prenant un point en argument et utilisant les coordonnées de ce dernier pour la création. Comment le compilateur sait quel constructeur appeler ?
9. Faire en sorte que l'appel à `System.out.println(point)` affiche les coordonnées du point comme ceci : (x, y) .



-
1. `main` est une méthode de la classe `Point`, donc a accès aux champs privés.
 2. `TestPoint.main()` n'est plus dans la classe `Point`, donc n'a pas accès aux champs privés de la classe `Point`. Assouplir l'accessibilité de `x` et `y` n'est pas une bonne idée. On peut créer des accesseurs (`getX` et `getY`), ou encore écrire une méthode d'affichage (`toString`, voir `exo3`).
 3. Cela permet de conserver une indépendance entre la classe qu'on développe (son implémentation) et les autres classes avec laquelle elle interagit.
 4. Un accesseur est une méthode qui permet, comme son nom l'indique, d'accéder indirectement en lecture ou en écriture à un champ (qui lui n'est pas accessible directement). Tant qu'on en a pas besoin, aucune raison de créer des setters. Les getters suffisent.
 5. En l'absence d'un constructeur explicitement défini, le compilateur en ajoute un (par défaut). Lorsqu'on définit au moins un constructeur explicite, celui par défaut n'est plus généré. Dans notre cas, le `main` avec `new Point()` ne compile plus. Il faut soit ajouter un second constructeur (surchargé), soit modifier le `main` en `new Point(0,0)` ;
 6. Si jamais les champs ont le même nom qu'une variable locale ou un paramètre, ce(tte) dernier(e) "masque" le champs. Ainsi, dans le constructeur `x = x ; //` ne fait rien d'autre que d'affecter la valeur du paramètre dans le paramètre... Il faut donc dans ce cas, utiliser la notion `this.x = x` ; Préférer cette notation systématiquement, au moins quand on est débutant.
 7. Il faut un champs `static` et la méthode doit aussi être `static`. Il vaut mieux faire le code dans le constructeur le + général et que les autres constructeurs y fassent appel par `this(...)`
 8. signature du constructeur



► Exercice 5. Égalité

On utilise la classe `Point` de l'exercice précédent.

1.

```
Point p1=new Point(1,2);
Point p2=p1;
Point p3=new Point(1,2);

System.out.println(p1==p2);
System.out.println(p1==p3);
```

Qu'affiche ce code ? Pourquoi ?

2. Écrire une méthode `isSameAs(Point)` renvoyant `true` si deux points ont les mêmes coordonnées.

3.

```
Point p1=new Point(1,2);
Point p2=p1;
Point p3=new Point(1,2);

ArrayList<Point> list = new ArrayList<>();
list.add(p1);
System.out.println(list.indexOf(p2));
System.out.println(list.indexOf(p3));
```

Quel est le problème ? Lire la doc d'`indexOf` (ou faire `control+clic gauche` sur `indexOf`) et indiquer quelle méthode est appelée. Modifier la classe `Point` pour résoudre le problème.

4. Que doit renvoyer `p1.equals(new String());` ?



-
- `==` teste l'égalité des références, pas l'égalité des objets référencés `System.out.println(p1==p2);`
`// true` : c'est la même référence au premier objet alloué qui est stockée dans `p1` et `p2`
`System.out.println(p1==p3);` `// false` : les deux objets alloués ont des champs qui ont la même valeur, mais ce sont deux objets différents.
 - avec ce code, on a bien `System.out.println(p1.isSameAs(p2));` `// true` `System.out.println(p1.isSameAs(p3));`
`// true`
 - Comme le dit la documentation, `indexOf(o)` returns the lowest index `i` such that `(o==null ? get(i)==null : o.equals(get(i)))`, or `-1` if there is no such index. Dans notre cas, cela signifie que `equals()` dit vrai entre `p1` et `p2`, et faux entre `p1` et `p3`. En effet, l'implémentation par défaut de `equals` est réalisée avec `==`.



► Exercice 6. Ligne brisée

On utilise toujours la classe `Point` de l'exercice précédent. On veut maintenant écrire une classe représentant une ligne brisée, c'est-à-dire une suite de points. La ligne brisée aura un nombre maximum de points défini à la création, mais pouvant varier d'une instance à une autre.

- On utilisera un tableau pour stocker les points d'une ligne brisée. Écrire le constructeur d'une ligne brisée.
- Écrire une méthode `add` ajoutant un point à la ligne brisée. Si on écrit pas de code supplémentaire, que se passe-t-il si on dépasse la capacité fixée ? Que faire ?

3. Écrire une méthode `pointCapacity()` et `nbPoints()` indiquant la capacité de la ligne brisée et le nombre de points actuellement sur la ligne.
4. Écrire une méthode `contains` indiquant si un point passé en argument est contenu dans la ligne brisée. Vous utiliserez pour cela une boucle **for each** et non une boucle classique.
5. Que se passe t-il si `null` est passé en argument à la méthode `contains`? Et si on a fait un `add(null)` avant? Regarder la documentation de `Objects.requireNonNull(o)`.
6. Soyez plus moderne et modifier la classe afin qu'elle utilise une `LinkedList` plutôt qu'un tableau (et ainsi ne plus avoir de limite sur sa taille). Que deviennent `pointCapacity`, `nbPoints` et `contains`?



-
1. Plus intéressant d'être défini à la construction de chaque `PolyLine`
 2. Leve une `ArrayIndexOutOfBoundsException`, faire une `illegalstate` spécifique.
 3. `false` si `p` est bien en argument du `equals` et pas l'appelé - marche si `p` est `null`, car `p` est un argument du `equals` - si `add(null)`, ajoute `null` dans le tableau, pose des problèmes par la suite dans le `contains()`. Utiliser `requireNonNull` pour éviter cela (leve une `NPE` au bon moment!)
 4. Supprimer `capacity`, `nbPoint` utilise `size` et `contains` le `contains` de `list`.

```

1 public class PolyLine {
2     private final Point[] points;
3     private int nb=0;
4
5     public PolyLine(int maxPoints) {
6         points = new Point[maxPoints];
7     }
8
9     public void add(Point p) {
10        if(points.length <= nb) throw new IllegalStateException("full");
11        points[nb++]=Objects.requireNonNull(p);
12    }
13
14    public int pointCapacity() {
15        return points.length;
16    }
17
18    public int nbPoints() {
19        return nb;
20    }
21
22    public boolean contains(Point p) {
23        for(Point p0:points) {
24            if(p0.equals(p)) return true;
25        }
26        return false;
27    }

```



► Exercice 7. Mutabilité et cercle

1. Ajouter une méthode `translate(dx, dy)` à `Point`. Quelles sont les différentes signatures et possibilités pour cette méthode ?
2. Écrire une classe `Circle`, défini comme étant un point (centre) et un rayon, ainsi que son constructeur.
3. Écrire le `toString`.
4. Écrire une méthode `translate(dx, dy)` qui translate un cercle.
- 5.

```
Point p=new Point(1,2);
Circle c=new Circle(p,1);

Circle c2=new Circle(p,2);
c2.translate(1,1);

System.out.println(c+" "+c2);
```

Quel est le problème ? Que faire pour l'éviter ?

6. Quel est le problème si on écrit une méthode `getCenter()` renvoyant le centre ? Pour y réfléchir, que fait le code suivant ?

```
Circle c=new Circle(new Point(1,2), 1);
c.getCenter().translate(1,1);
System.out.println(c);
```

Modifier pour que cela soit correct.

7. Ajouter une méthode `surface()` et l'ajouter dans l'affichage du cercle.
8. Ecrire la méthode `equals` d'un `Circle`.
9. Créer une méthode `contains(Point p)` indiquant si le point `p` est contenu dans le cercle (indice : utiliser pythagore).
10. Créer la méthode `contains(Point p, Circle...circles)` qui renvoie vrai si le point est dans un des cercles. Doit-on en faire une méthode statique ?



-
1. Mutable ou non mutable ? Modifier les champs (non final) vs renvoyer un nouveau `Point`.
 2. .
 3. .
 4. si mutable, on peut modifier un cercle a partir d'un autre !
 5. si on fait pas de copie défensive, on peut modifier un cercle à partir d'un getter !

```
1 public class Circle {
2     private final Point center;
3     private final double r;
4
5     public Circle(Point p, double r) {
6         //this.p = p;
7         this.center = new Point(p); //copie defensive
8         this.r = r;
9     }
10 }
```

```

11 public Circle translate(double dx, double dy) {
12     return new Circle(center.translate(dx, dy), r);
13 }
14
15 @Override
16 public String toString() {
17     return center.toString() + " " + r + " " + surface();
18 }
19
20 public Point getCenter() {
21     //return p;
22     //copie defensive
23     return new Point(center);
24 }
25
26 @Override
27 public boolean equals(Object obj) {
28     if(obj==this)return true;
29     if(!(obj instanceof Circle)) return false;
30     Circle c = (Circle)obj;
31     return c.center.equals(center) && r==c.r;
32 }
33
34 public double surface() {
35     //pi.r^2
36     return Math.PI*r*r;
37 }
38
39 /*distance au carré*/
40 private double squareDistance(Point p) {
41     double dx = p.getX() - center.getX();
42     double dy = p.getY() - center.getY();
43
44     return dx * dx + dy * dy;
45 }
46
47 public boolean contains(Point p) {
48     //pythagore
49     return squareDistance(p) <= r*r;
50 }
51
52 public static boolean contains(Point p, Circle...circles) {
53     for(Circle c: circles) {
54         if(c.contains(p))return true;
55     }
56     return false;
57 }

```

----- ✂

► Exercice 8. Anneaux

On veut définir maintenant un anneau : un cercle dont un cercle interne a été supprimé.

1. Est-ce intéressant de faire de l'héritage ici ?
2. Écrire une classe `Ring`, prenant en argument un centre, un rayon et un rayon interne (qui doit être inférieur au rayon).

3. Écrire la méthode `equals`.
4. On veut afficher un anneau avec son centre, son rayon et son rayon interne. Quel est le problème si on fait `System.out.println(ring)`; sans code supplémentaire? Le corriger.
5. Écrire une méthode `contains(Point)` en évitant d'allouer des objets ou de dupliquer du code.
6. Écrire la méthode `contains(Point p, Ring...rings)`.



-
1. Oui, le faire
 2. faire le super
 3. Appeler le super
 4. Redéfinir!
 5. sioux
 6. Ne pas le faire, fait par héritage et appel le bon contains!

```

1 public class Ring extends Circle {
2     private final double intR;
3
4     public Ring(Point center, double r, double intR) {
5         super(center, r);
6         if(intR>=r) throw new IllegalArgumentException("internal must be smaller..");
7         this.intR = intR;
8     }
9
10    @Override
11    public boolean equals(Object o) {
12        if(!(o instanceof Ring)) {
13            return false;
14        }
15        Ring r = (Ring)o;
16        return intR == r.intR &&
17            super.equals(r);
18    }
19
20    //doit redefinir toString pour pas utiliser celui de Circle !
21    @Override
22    public String toString() {
23        return super.toString() + " " + intR;
24        //attention, fait appel au surface de Circle !! donc faux.
25    }
26
27    @Override
28    public boolean contains(Point p) {
29        //allocate a new circle
30        // Circle internal = new Circle(super.getCenter(), internRadius);
31        // if(internal.contains(p)) return false; //partie evide
32        // return true;
33
34        //no allocations
35        //squareDistance private=> protected
36
37        // mauvaise solution, on calcule deux fois la distance au carre
38        // if (!super.contains(p)) {
39        //     return false; //meme pas dans le cercle
40        // }
41        // return squareDistance(p) >= innerRadius*innerRadius;
42

```

```
43     double squareDistance = squareDistance(p);
44     return squareDistance <= getR()*getR() &&
45         squareDistance >= intrR*intrR;
46 }
47
48 public static boolean contains(Point p, Ring...rings) {
49     //le contains de circle va appeller le contains sur les ring car ce sont des ring..
50     // et des Ring[] sont des Cicle[] mais le contains sur un Circle
51     // appel bien celui de Ring si c'est effectivement un ring.
52     return Circle.contains(p, rings);
53
54     // ou mieux, enlever cette méthode car on récupère celle
55     // de Circle par héritage
56 }
```

----- ✂