*Vous devez **impérativement modifier vos codes** en fonction des remarques ci-dessous.*
N'oubliez pas les Remarques Globales (RG) et lisez les commentaires des autres

Explication de ce qui est testé par le tableau (KO, OK):

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
Constructeurs bien organisés Inits en cascade des classes parents	Définition correcte KO si manque les méthodes d'accès au nb de sommets et à un sommet	Classe abstraite Bien conçue pour la liste circu. Ex: ne doit pas stocker de tableau de Point Evite le copier-coller.	Gestion de la Rotation OK Capable de comparer entre des implèm différentes (y compris des implémentations futures)	Implém correcte OK-si méthodes inefficaces notamment getUnSommet	Bonne implém OK-si aurait pu être abstrait ou si implem juste mais améliorable	Sous-classes TriangleTab CarreTab... Calcul spécialisés (perimètre, surface) Utilisation de vars instances propres. OK ½ si pas TriangleLstCir,...	Bien rédigée compilée

Remarques globales (en **gras** les erreurs très graves) :

1. Utiliser une constante symbolique EPSILON (public static final int) dans la classe Point
2. **Pas d'interface Polygone ! C'est un des buts du TD**
3. **Interface Polygone : pas de méthodes pour obtenir le nombre de sommets, ni un sommet donné (ni tous). Mettre un getSommets() qui renvoie un tableau est un mauvais choix qui n'est valable que pour PolygoneTab, pas pour PolygoneLstCirc…**
4. **Interface Polygone** : pas assez de fonctionnalités dans l'interface.
5. **Pas d'itérateur pour passer en revue les sommets d'un polygone**
6. Les classes implémentant un polygone ont un itérateur mais ce n'est pas dit dans la définition de l'interface Polygone : du ce fait pas de foreach possible ! Cette interface doit dire qu'elle étend Iterable<Point>.
7. **Pas de classe abstraite. Donc pas de factorisation de code possible (=> copier-coller).**
8. **Méthode equals de polygone : ne tient pas compte de la rotation possible (comparaison 2 à 2) !**
9. **Méthode equals de polygone : ne tient pas compte de l'ordre des points !**
10. **Méthode equals dans classe abstraite ne gère pas bien la comparaison entre différentes implémentations (y compris futures). Il suffit de tester si c'est une instance de Polygone et de caster en Polygone. C'est tout !**
11. **La classe abstraite ne doit pas stocker les points sous forme de tableau : sinon aucun intérêt d'avoir une représentation en liste circulaire ! Mauvaise abstraction.**
12. La classe abstraite ne devrait pas définir surface qui retourne -1. C'est à l'implémentation concrète de prendre cette responsabilité (ça peut être acceptable mais ne force plus les classes concrètes à se poser la question).
13. L'implémentation liste circulaire doit avoir un getUnSommet() optimisé (il faut éviter de repartir depuis le début de la liste à chaque appel). Sinon sera inutilisable en pratique.
14. **Méthode main() insuffisante : pas assez de tests de cas d'erreurs**

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK–	OK	OK–	OK–	KO	KO	KO	OK

RG : 10,12,13
Classe abstraite : attribut taille devrait être private ou protected ! Par ailleurs cet attribut n'est jamais utilisé !
Le code du constructeur semble inachevé : test sur la taille ne faisant rien, boucle calculant « compteur » mais on n'en fait rien… Je ne vois pas pourquoi il reçoit un tableau de Points alors.
Dans perimetre() le code suivant ne sert à rien. Si on appelle perimetre() dans la classe abstraite c'est que c'est un polygone concret. Nul besoin de le caster.

```
PolygoneAbs p = null;
if(this instanceof  PolygoneTab){
   p = (PolygoneTab) this;
}
if(this instanceof  PolygoneLstCirc){
   p = (PolygoneLstCirc) this;
}
```

Code de equals() : le même que ci-dessus code apparaît dans equals(). Ici on ne sait pas si c'est un Polygone car on reçoit un Object. Il faut donc tester ; mais tous ces tests peuvent se simplifier en :

```
if (o == null || !(o  instanceof  Polygone)
      return false
Polygone p = (Polygone) o;
```

car on peut utiliser l'interface comme nom de type et comme nom pour un cast.
Les variables j et compteur ont la même valeur. A simplifier.
NB : ce n'était pas une très bonne idée de tout faire dans la même boucle : on décompose en 2 boucles consécutives : 1) boucle qui cherche premier point le j dans polygone2 et 2) boucle qui compare 2 à 2 une fois trouvé le premier.
Le count%p.getNbSommets() devrait être inutile puisque le getUnSommet fait le %. Je regarde le code (oups ce n'est pas le cas pour PolygoneTab qui, au passage fait n'importe quoi). Grrr. Code pas testé correctement.
Pour en finir avec le equals : vous devriez simplifier de vous-même ce code :

```
if (!identique)
      return false;      se simplifie en          return identique ;
return true;
```

Idem pour dautres codes (ex : memeReel dans Point)
Code de l'itérateur faux : le indiceCourant++ dans le next fait que le premier point retourné sera en fait celui d'indice 1 et pas d'indice 0 (je comprends du coup le code faux de getUnSommet() de PolygoneTab, mais c'est une mauvaise rustine). Il fallait incrémenter après le getUnSommet(), ce qui peut s'écrire : return getUnSommet(indiceCourant++);
Implem liste circulaire : le getNbSommets() est irréaliste : il faut garder ce nombre de sommets dans une variable d'instance sinon c'est trop couteux à calculer chaque fois. De plus le code est mal écrit (le test || compteur == 0 ne sert à rien, il est toujours faux vu que compteur est incrémenté juste avant il ne vaudra jamais 0).
Bilan : de bonnes choses mais manque d'harmonisation du code et de finition

BARBIER Thomas / KERVAREC Samuel 14,5/20

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK–	OK	OK–	KO	OK–	OK	OK––	OK

RG : 1,8,9,10,12,13
Classe Point : déclarer une constante EPSILON
PolygoneAbstrait : retirer ce constructeur qui ne sert à rien.
J'aurais préféré que la classe abstraite ait pour attribut le nb de sommets (qui sera fourni quelque soit l'implem) et donc avoir un constructeur recevant ce nb de sommets. Elle fournira alors aussi le getter getNbSommets() - cela évite aux clases d'implem d'avoir à gérer cela (factorisation).
equals : mauvais tests de départ (type + cast) :

```
if (obj == null || !(obj instanceof Polygone))
    return false;
Polygone polygone = (Polygone) obj;
```

equals() algo de comparaison faux, il faut tenir compte de l'ordre des points. Il faut d'abord chercher le « point de départ » (où se trouve le 1er point du polygone 1 dans le polygone 2) et ensuite faire une comparaison 2 à 2, dans l'ordre (circulaire dans le polygone 2 grâce au modulo). Il faut 2 boucles consécutives.
Cf code dans remarque au groupe DEKKAR Younès / MADI-RACHIDI Armel
Iterateur : enlever l'attribut public boolean hasNext ! ne sert à rien et induit en erreur avec le même nom que la méthode hasNext().
PolygoneTab ne devrait pas fournir un getSommets() qui est propre à l'implem.
RectangleTab : t n'est pas un attribut, c'est une var locale de surface() !!!
Pourquoi si différent de RectangleLstCirc ? Doivent être les mêmes !
Dans classes TriangleXXX attributs devraient être en private
Listeolygone : attribut déclare comme List<PolygoneAbstrait> listePolygone, cela implique qu'on sait que c'est implémenté via une classe abstraite (or ici on est utilisateur de polygones). Il faut simplement déclarer l'attribut avec List<Polygone> listePolygone.
Javadoc : attention relecture (cf Polygone, typos, ou return manquant dans Point)
Bilan : manque juste un peu d'attention et de finition (ex : private pour les attributs !!!).

BENABOU Yoel / GUENEGO Maxime 16,5/20

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK–	OK	OK	KO	OK	OK	OK–	OK

RG : 12
Att convention : nom de package en minuscule.
Rq mineure : le nom de méthode getTaillePolygone() n'est pas top. D'abord le mot « Polygone » ne sert à rien puisque il s'agit des fonctionnalités d'un polygone (sinon le mettre partout, ex surfacePolygone() mais ce n'est pas la convention). Ensuite la « taille » d'un polygone, je ne sais pas trop ce que c'est (peut laisser penser à son périmètre ?). Enfin, ce n'est pas cohérent avec le getUnSommet(). Donc il me semble que ça devrait être getNbSommets(). Pour info on parle aussi de « l'ordre d'un polygone » pour son nombre de côtés (= nombre de sommets). Bref, les noms ont de l'importance.
Classe Point : l'avantage des objets non-mutables c'est, que ne changeant pas, on n'est pas obligé de recalculer toujours la même chose. Ainsi le toString renverra toujours la même valeur ; pas la peine de la recréer à chaque fois. Voici un code pour optimiser cela (mise en cache grâce à une variable d'instance str) :

```
private String str = null ;
   public String toString(){
   if (str == null) // ou votre StringBuilder str… str = message.toString();
       str = "<" + this.getX() + "," + this.getY() + ">";
   return str;
}
```

NB : on pourrait faire de même avec hashCode(),...
Polygone abstrait : en toute logique, le constructeur ne devrait recevoir que la taille du polygone et pas tout le tableau (qui ne lui sert pas et qui oblige toute implémentation future à fournir un tableau). C'est aux classes concrètes de l'appeler avec la taille : super(tab.length);
equals() : mauvais tests de type et cast (car reçoit un Object). Il suffit de vérifier que c'est un Polygone et de le caster en Polygone. On peut utiliser le nom de l'interface comme type :

```
if (objet == null || !(objet instanceof Polygone))
     return false;
Polygone polygone = (Polygone) objet;
```

equals() faux.
Cf code simple dans remarque au groupe DEKKAR Younès / MADI-RACHIDI Armel.
getUnSOmmet() dans lst circ : TB mais quel dommage d'avoir mis en public l'accès au nœud en cache (getUnNoeuCourant()). A quoi pourrait bien servir ce Nœud à l'utilisateur ??? Idem pour getIndiceCourant().
Même erreur dans TriangleXXX avec getTableauDistances (c'est un tableau interne pour accélérer les calculs, n'apporte pas de fonctionnalités). De manière générale, il faut bien réfléchir lorsque une classe d'implémentation (ex : TriangleTab) offre (i.e. public) une fonctionnalité non prévue pas l'interface qu'il implémente (c-a-d qu'il n'y a pas de @override devant la méthode) ; ça devrait être exceptionnel et justifié.
ListePolygone : déclarer la liste avec les interfaces ; au lieu de

```
private ArrayList<PolygoneAbs> listPolygone;
```

il faut

```
private List<Polygone> listPolygone;
```

Bilan : très bon travail d'ensemble. On peut encore améliorer…

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK–	KO	OK	OK	OK–	OK–

RG : 8,9,13
Classe Point, c'est quoi cet attribut Point b ? a supprimer
Le code de memeReel se simplifie en return Math.abs(x - y) <= EPSILON;
Interface Polygone : dans une interface on n'écrit pas les public abstract (c'est par défaut)
Polygone Abstrait : attribut int nbSommets; devrait être protégé (private ou protected).
perimetre() : comme le getUnSommet() s'engage a faire un %nbSommets, vous pouvez simplifier ce code (pas la peine de gérer la distance du dernier au premier en dehors de la boucle).

```
double sum = 0;
for (int i = 0; i < nbSommets; i++)
   sum += this.getUnSommet(i).distance(this.getUnSommet(i + 1));
return sum;
```

Pas de equals dans la classe abstraite ? dommage (factorisation), il va falloir être sûr que les codes de chaque classe d'implémentation fait bien la même chose et sans bug. Si on corrige un bug à un endroit il faudra le faire pour toutes les autres implémentations. Aie.
Je regarde le code equals de PolygoneTab : faux. Je regarde celui de PolygoneLstCirc : il n'y en a pas; donc utilise le equals par defaut de Object (comparaison de pointeurs). Incohérent.
L'incohérence apparaît aussi dans Rectangle (RectangleTab garde les distances en variables d'instances (ce qui est TB) mais pas RectangleLstCirc – pourquoi ?). Idem pour les carrés et les triangles. A reprendre et harmoniser. (Ce devraient être les mêmes codes).
Javadoc incomplète
Bilan : de bonnes choses, du travail et des progrès. Il faut être attentif et bien relire son code (maintenant il ne devrait plus y avoir d'attribut inutilisés ou sans private/protected,...).

CHINOUNA Morchid / NGO Stéphane 13,5/20

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK	KO	OK	OK	OK--	KO

RG : 6,9,10,12,13
Classe Point : la constante EPSILON devrait être en public.
Classe abstraite : si elle garde le nombre de sommets (appelé ici taillePoly), c'est à elle de fournir le getter getNbSommets() ! Pas aux classes d'implem.
equals(). mauvais tests d'instance + cast, devrait être :

```
if (obj == null || !(obj instanceof Polygone))
    return false;
Polygone polygone = (Polygone) obj;
```

je ne comprends pas l'algo, ça commence bien (on cherche l'indice « index » où se trouve le premier point dans le second polygone) mais ensuite index n'est pas utilisé et la double boucle laisse penser à une comparaison sans tenir compte de l'ordre. En fait il suffit de comparer les points 2 à 2 (Attention au test == qui est faux chez vous, il faut utiliser equals). La suite était pourtant simple :

```
for (int i = 1; i < this.getNbSommets(); i++) {
    if (!(this.getUnSommet(i).equals(p.getUnSommet(index + i)))
        return false;
return true;
```

Calcul du perimetre : l'utilisation de 2 iterateurs n'est pas la plus simple ; je préfère le code commenté mais attention ne marche que si getUnSommet fait un modulo (et il ne le fait pas, cf ci-dessous).
PolygoneTab : c'est bien d'avoir mis le getter de sommets en protected car on ne doit pas dévoiler l'implem. Malheureusement ça ne sert à rien car l'attribut sommets est public !!!
getUnSommet() devrait faire un modulo nbsommets pour « tourner en rond » (et éviter les débordements). Ca simplifie le reste du code (y compris celui ci-dessus pour equals).
LstCirc : getUnSommet() incohérent (pourquoi < indice + 1 ? c'est < indice sinon retourne le point suivant).
Liste Polygone : attribut faux ArrayList<PolygoneTab> listPoly ne permet pas de stocker des polygones en listes circulaires ! Ca doit être : List<Polygone> listPoly et la construction doit se faire avec new ArrayList<>() ; (votre new ArrayList() perd les generic)
javadoc ? Au moins celle l'interface (c'est la plus importante pour l'utilisateur) !!!
Bilan : De bonnes choses mais c'est un travail « en chantier ». Dommage. Il faut être encore plus exigeant avec votre code.

DEKKAR Younès / MADI-RACHIDI Armel 15,5/20

constr / héritage	Interface	Abstrac-tion	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK	OK	OK	OK	OK	OK

RG : 6,10,12,13
Nom du package « point » mal choisi.
Classe Point : la constante EPSILON devrait être en public.
calcul de distance : le abs() ne sert à rien, une racine (sqrt) est toujours positive
je ne comprends pas l'utilité de distanceX() et distanceY() si elles ne sont pas utilisées par distance() !
Interface Polygone : les « public » devant les méthodes sont optionnels.
classe abstraite :
equals(). mauvais tests d'instance + cast, devrait être :

```
if (obj == null || !(obj instanceof Polygone))
    return false;
Polygone polygone = (Polygone) obj;
```

l'algo de equals est bon !
On peut le simplifier : ainsi le test if (!getUnSommet(0).equals(poly.getUnSommet(0))) {
ne sert à rien (à supprimer), car la boucle qui cherche la correspondance du premier point s'arrêtera alors dès le premier test tout simplement. Le booléen trouvé peut être évité. Normalement getUnSommet fait un modulo donc pas besoin de le faire ici (je regarde PolygoneTab : il le fait (bien), je regarde PolygoneLstCirc : il ne le fait pas :-(pourquoi ?).
Voici une simplification pour equals :

```
int deb = 0;
while (deb<getNbSommets() && !getUnSommet(0).equals(poly.getUnSommet(deb)))
    deb++;
if (deb >=getNbSommets())
   return false;
for (int i = 0; i < getNbSommets(); i++) {
   if ( !getUnSommet(i).equals(poly.getUnSommet(i + deb)))
      return false ;
}
return true;
```

perimetre() se simplifie grace au fait que getUnSommet() fait le modulo.
ListePolygone : la declaration private ArrayList<PolygoneTab> liste = new ArrayList();
est fausse (ne marche pas pour des liste circulaires). Devrait être
private List<Polygone> liste = new ArrayList<>();
Bilan : de très bonnes choses et des maladresses. Il faut plus relire vos codes.

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK	OK	OK	KO	OK–	KO

RG : 10,14
Point : EPSILON = Math.pow(10, -10); s'écrit plus simplement EPSILON = 1e-10;
classe abstraite :
Iterateur : ne marche pas 2 fois d'affilées car indiceCourant est une variable d'instance de PolygoneAbs.
c'est bien d'avoir fait surface() mais il manque dans la doc une expli de l'algo (pas si évident) ou à defaut une référence d'où vous avez pris l'algo (ce qui est OK). Par contre je ne vois pas l'utilité des tableaux abscisses et ordonnées, il suffit d'utiliser à la place getUnSommet(i).getX() et getY() (économie de place). Par ailleurs ca éviterait d'allouer nbsommets +1 case (et son init) puisque getUnSommet() fait le modulo. A simplifier
equals () : le code de test+cast au début est pas mal, on peut le simplifier en testant Polygone plutôt que PolygoneAbs (test + cast).
Par contre l'algo de comparaison est faux
Lors d'un new Iterateur il faut qu'il soit remis à zéro (sinon garde la valeur du dernier parcours avec l'itérateur et est donc > nbSommets). Il faut mettre cette variable en tant que variable d'instance de Itr.
Pour le next() ce n'est pas getUnSommet(++indiceCourant)
mais getUnSommet(indiceCourant++). Code pas testé !
PolygoneTab : pas de getter public pour les sommets ! devoile l'implem (mettre protected).
PolygoneLstCirc : c'est bien d'avoir optimisé le getUnSommet() ! J'aurais juste mis le modulo sur indiceCourant plutôt que sur le test (à la langue cet indice va déborder des 32 bits). Voici : for (; indiceCourant != numSommet; indiceCourant=(indiceCourant+1)% getNbSommets())
Triangle : le code de détection du type (scalène,...) est maladroit. Rq : le test pour équilatéral demande 3 tests avec memeReel (ca ne devient plus transitif, à cause du EPSILON).
Voici mieux :

```
int cmp = 0
cmp += Point.memeReel(a, b) ? 1 : 0 ;
cmp += Point.memeReel(b, c) ? 1 : 0 ;
cmp += Point.memeReel(a, c) ? 1 : 0 ;
if (cmp == 3)
      return ("Triangle équilatéral");
if(cmp == 0)
      return "Triangle scalène";
return "Triangle isocèle";
```

LitePolygone : private ArrayList<Polygone> polygones; c'est bien d'avoir utilisé <Polygone> pour le generic (l'interface). Pour ArrayList c'est pareil, utiliser l'interface (c'est List):
private List<Polygone> polygones;
javadoc de l'interface ???!!! La plus importante !
Bilan : de très bonnes choses (surface de polygone) mais des oublis et maladresses. Plus d'exigence avec vos codes. Et surtout... plus de tests !

GALLART Julien 15,5/20

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK	OK–	OK	KO	OK	OK

RG : 6,10,12
Classe Point : EPSILON devrait être public (aucun risque c'est une constante)
Att : 1*10^(-9) ne fait pas ce que tu crois car ^ est un XOR bit à bit. Soit -3 !
Il faut écrire EPSILON = 1e-9;
Polygone : doit dire qu'elle implémente l'itérateur sinon pas de foreach
classe abstraite :
equals(). mauvais tests d'instance + cast, devrait être :

```
if (obj == null || !(obj instanceof Polygone))
    return false;
Polygone polygone = (Polygone) obj;
```

l'algo de equals est trop compliqué (même si l'approche est bonne). Lorsque le premier point est trouvé dans le second polygone il faut sortir de la boucle et en faire une autre après pour comparer 2 à 2. Grâce au modulo pas besoin de décomposer cette comparaison en 2 parcours comme tu le fais. Il faut écrire simlplement !
Voir code simple dans remarques DEKKAR Younès / MADI-RACHIDI Armel
Iterateur : ne marche pas si on l'appelle 2 fois car la variable « indice » est une variable d'instance de PolygoneAbs. Donc mise à zéro une seule fois à la création du polygone. Lorsqu'on a fini un parcours (iterateur), elle vaut nbSommets. A la création d'un nouvel itérateur elle restera à cette valeur et hasNext() retournera toujours faux. La remettre à zéro à la création de l'Itr (en ajoutant un constructeur à Itr ou au retour du new Itr() dans iterator()) n'est pas la meilleure solution. Dans une appli multi-threadé on ne pourrait pas utiliser 2 itérateurs en parallèles sur le même polygone (ce qui est dommage) puisqu'ils se partageraient le même « indice ». La bonne solution consiste à déplacer la définition de l'attribut « indice » pour le mettre dans la classe Itr. En devenant une variable d'instance de Itr(), indice devient propre à chaque itérateur, il est initialisé à 0 à chaque new et il n'y a plus de pb de multi-threading.
Toujours dans l'itérateur, tu as une erreur dans next() car le ++indice est fait avant (donc au premier appel tu retournes le second point en fait). Par ailleurs ton test hasNext() ne sert à rien (il semble repartir à 0 mais le modulo de getUnSommet() fait pareil !). Voici le bon code :

```
public Point next() {
    return getUnSommet(indice++);
}
```

PolygoneLstCirc : le getUnSommet() ne fait pas le modulo (alors que c'est fait pour PolygoneTab). Incohérent. Si on donne un très grand indice, ça « tournera en rond » sur la liste circ pour rien. Ajouter ici aussi, en début : indicePoint %= this.getNbSommets();
Classes Triangle : c'est bien de fournir des méthodes telles que public boolean isIsocele(). Il faut donc qu'elles soient publiques. Mais comment l'utilisateur va appeler ces méthodes ? Il faut caster, en quoi ? TriangleTab, TriangleLstCirc ? et s'il ne sait pas ? C'était l'occasion de créer une interface Triangle... (qui étend Polygone) et de dire que TriangleTab et TriangleLstCirc implémentaient cette interface, ainsi :
public class TriangleTab extends PolygoneTab implements Triangle ...
Bilan : de très bonnes choses, encore perfectible.

GODEFERT Lucie / SIBAUD Nicolas 16,5/20

constr / héritage	Interface	Abstraction	Polygone Equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK	OK–	OK	OK	OK	OK–

RG : 12,13
Classe Point : EPSILON = 0.000000001 s'écrit plus simplement EPSILON = 1e-9;
equals() : la variable objet O doit s'appeler 'o' (convention var comment par minuscule).
Interface polygone : la doc de surface dit « non-implémentée pour l'instant ». Mauvais commentaire ! l'interface donne des fonctionnalités, elle ne devrait pas parler d'implem. Ce commentaire vieillira mal (que se passe-t-il si c'est implémenté par les PolygoneTab et pas par PolygoneLstCirc ?). A la rigueur mettre un commentaire du genre « cette méthode n'est pas toujours implémentée » et au @return la surface (ou -1 si pas implémenté).
Tant qu'on parle de doc : la doc de getUnSommet(int indicePoint) devrait dire que l'indice est numéroté à partir de 0.
classe Abstraite :
equals(). mauvais tests d'instance + cast, devrait être :

```
if (obj == null || !(obj instanceof Polygone))
    return false;
Polygone polygone = (Polygone) obj;
```

l'algo de comparaison est « juste » mais tellement mal écrit, si compliqué. Pourquoi faire 2 boucles imbriquées alors qu'il s'agit de 2 boucles consécutive (1 pour trouver la correspondance du premier point dans le polygone 2 et l'autre pour comparer 2 à 2) ? Pourquoi ces 2 boucles pour la comparaison 2 à 2 (le modulo fait tout).
cf algo simple donné pour le groupe DEKKAR Younès / MADI-RACHIDI Armel.
Classes Triangle : c'est bien de fournir des méthodes telles que public boolean isIsocele(). Il faut donc qu'elles soient publiques. Mais comment l'utilisateur va appeler ces méthodes ? Il faut caster, en quoi TriangleTab, TriangleLstCirc ? et s'il ne sait pas ? C'était l'occasion de créer une interface Triangle... (qui étend Polygone) et de dire que TriangleTab et TriangleLstCirc implémentaient cette interface, ainsi :
public class TriangleTab extends PolygoneTab implements Triangle ...
Bilan : de très bonnes choses, encore perfectible.

GURGEV Momchil 13,5/20

constr / héritage	Interface	Abstrac-tion	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	KO	OK-	KO	OK	OK	OK	OK-

RG : 3,9,12
Classe Point : EPSILON = 0.000000001 s'écrit plus simplement EPSILON = 1e-9;
equals() : cette méthode de clase (static) avec 2 arguments n'est pas du tout ce que l'on utilise en Java. La norme est d'utiliser la méthode d'instance avec un argument, qui (par héritage de Object) doit être du type Object ! Cf cours.
ainsi si on doit comparer 2 points p1 et p2 on écrit p1.equals(p2). Ici il faudrait utiliser Point.equals(p1,p2). Je regarde si elle est ainsi utilisée dans le TD (dans le equals de PolygoneAbs) et je vois ... qu'elle n'est pas utilisée, c'est bien le p1.equals(p2). Ne marchera pas ! (On appellera le equals hérité de Object qui fait une comparaison de pointeurs).
Polygone : manque les accesseurs (getNbSommets() et getUnSommet()) !
javadoc absente - celle de l'interface est la plus importante !
classe Abstraite : Variable d'instace int nbrSommets = 0; pas protégée ! (mettre private ou protected). Constructeur : exceptions à revoir.
perimetre() : 3 itérateurs ! Ici une boucle avec 2 getUnSommet() aurait été bien plus simple et plus efficace.
equals(). mauvais tests d'instance + cast, devrait être :

```
if (obj == null || !(obj instanceof Polygone))
    return false;
Polygone polygone = (Polygone) obj;
```

l'algo de comparaison est faux (il faut tenir compte de l'ordre des points 2 à 2).
Dommage de ne pas avoir factorisé l'itérateur dans la classe abstraite, ça aurait évité de devoir l'implémenter n fois pour n implems (ici n=2). Avec un getUnSommet() c'était facile à faire.
TriangleXXX : il ne faut pas comparer les distances avec == (ex : ab == bc) mais avec Point.memeReel(ab,bc) !
Bilan : de bonnes choses et des capacités à très bien faire. Go !

LECOUPLE Valentin / RAMANE Pierre-Antoine 16/20

constr / héritage	Interface	Abstrac-tion	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK	OK	OK	OK	OK–	OK

RG : 12,13
Polygone : getUnPoint() devrait plutôt s'appeler getUnSommet() (cohérent avec getNbSommet()).
classe abstraite : elle contient la variable d'instance nbSommet (TB, ça factorise) mais alors je ne comprends pas pourquoi son constructeur reçoit un tableau (alors que seul le nb de sommets est nécessaire, i.e un int). Cela force toutes les implémentations (y compris les futures) à disposer d'un tableau alors qu'il suffit qu'elles passent le nb de sommets. A modifier.
perimetre() : calcul faux, manque la dernière distance (entre dernier sommet et premier). Il suffit de modifier la boucle en for (int i = 0; i < getNbSommet(); i++) et ça va marcher si getPoint (getUnSommet()) fait le modulo dont on a parlé en cours. Je regarde, aie il ne le fait pas, dommage : il suffit d'ajouter indexPoint %= getNbsommets() au debut des méthodes getUnPoint() de PolygoneTab et PolygoneLstCirc et c'est bon.
equals() : le test de validité de type n'est pas cohérent avec le cast (qui est bon !). Faire :
if (obj == null || !(obj instanceof Polygone))
l'algo de comparaison de equals est bon (même si on peut le simplifier), ainsi le if suivant ne sert à rien si tu fais faire le module à getPoint() comme explique ci-dessus.

```
if (indexSommetCommun == p.getNbSommet()) {
   indexSommetCommun = 0;
}
```

Sinon, il vaut mieux utiliser le modulo (évite un if) lors de l'incrémentation :
indexSommetCommun++; devient
indexSommetCommun = (indexSommetCommun + 1) % getNbSommets();
Voir code simple dans le groupe
Dommage que les classe RectangleXXX, TriangleXXX n'utilisent pas de variable d'instances pour éviter les recalculs (mais c'est bien fait pour CarreXXX),
TriangleXXX : dans test équilatéral if (Point.memeReel(c1, c2) && Point.memeReel(c1, c3)) pas suffisant, à EPSILON près l'égalité n'est plus transitive, il faut aussi tester c2 et c3.
ListePolygone : private ArrayList<PolygoneAbstract> liste; doit en fait être déclarée comme
private List<Polygone> liste; (on utilise les interfaces pour les declarations, List et Polygone). Pour Polygone c'est particulièrement important car on est ici du côté utilisateur de la librairie polygones et on n'est pas censé savoir qu'il y a une classe abstraite (c'est une technique d'implem pour factoriser du code dans notre cas).
Bilan : très bon travail d'ensemble. Bien relire les codes et s'interroger (par exemple sur le -1 à la boucle de perimetre()).

NOR Senaroith / TOQUE Yann 15,5/20

constr / héritage	Interface	Abstraction	Polygone equals	Implem liste circ.	itérateur	classes spécialisées	javadoc
OK	OK	OK–	KO	OK	OK	OK–	OK–

RG : 9,12
Polygone : javadoc att de bien commenter les @return (ne pas mettre le type, javadoc le déduit).
PolygoneAbstract : pas besoin de déclarer getNbSommets() et getUnSommet() en asbtract puisqu'elles sont dans l'interface.
NB : cette classe pourrait avoir pour attribut le nb de sommets (car c'est commun à toutes les implems) et elle donnerait alors le getNbSommets().
perimetre() : ok mais le modulo dans this.getUnSommet((i + 1) % nbSomm) devrait être inutile puisqu'on avait convenu que getUnSommet() le faisait. Je vérifie , c'est le cas pour PolygoneListCirc (bien) mais pas pour PolygoneTab ! Il ne faut pas laisser ce genre d'incohérence !
equals() : ici aussi on peut de passer du modulo.
att : algo de comparaison faux : il faut respecter l'ordre des points !
Pourquoi ne pas avoir factorisé toString() dans cette classe ? Regardez son code dans PolygoneTab et PolygoneListCirc : idem.
PolygoneListCirc : c'est bien d'avoir optimisé le getUnSommet() !
TriangleXXX : utilisez des variables locales pour ne pas refaire toujours les mêmes calculs de distances dans une même méthode.
c'est bien de fournir des méthodes telles que public boolean isIsocele(). Il faut donc qu'elles soient publiques. Mais comment l'utilisateur va appeler ces méthodes ? Il faut caster, en quoi TriangleTab, TriangleLstCirc ? et s'il ne sait pas ? C'était l'occasion de créer une interface Triangle... (qui étend Polygone) et de dire que TriangleTab et TriangleLstCirc implémentaient cette interface, ainsi :
public class TriangleTab extends PolygoneTab implements Triangle ...
Bilan : de très bonnes choses et du progrès ! Encore des améliorations possibles. Il faut persévérer.

TRUONG Anh Quoc Thong 0/20

Travail non fait.