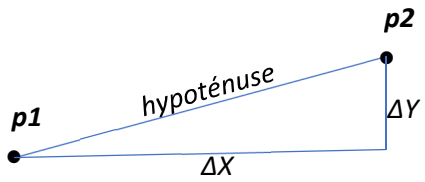
Interactions des objets géométriques

La distance entre 2 points

Supposons p1 et p2 deux objets de la classe **Point**.

⇨ *Comment calculer la distance entre ces deux points ?*



Dans ce dessin, nous voyons que la distance entre ***p1*** et ***p2*** correspond à l'*hypoténuse* d'un triangle droit. Le côté *ΔX* est égal à la différence des coordonnées x des deux points ***p1*** et ***p2*** et le côté *ΔY* est égal à la différence des coordonnées y des deux points ***p1*** et ***p2***, donc :

ΔX = $|$***p2.x*** – ***p1.x***$|$, ΔY = $|$***p2.y*** – ***p1.y***$|$, et *hypoténuse* = $\sqrt{(\Delta X)^2 + (\Delta Y)^2}$.

En java, pour une méthode ***distance(…)*** qui calcule la distance entre deux points, il faut écrire :

```
public double distance(Point p1, Point p2) {
    return Math.sqrt(Math.pow(p2.x-p1.x,2)+Math.pow(p2.y-p1.y,2));
}
```

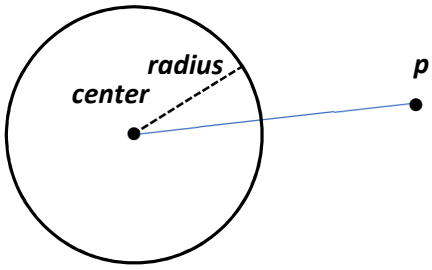
ou encore, pour avoir une méthode qui donne un entier arrondi comme résultat :

```
public int roundedDistance(Point p1, Point p2) {
    return (int) Math.round(
        Math.sqrt(Math.pow(p2.x-p1.x,2)+Math.pow(p2.y-p1.y,2)) );
}
```

Un point se trouve sur un disque

Supposons une classe **Disk** qui possède les attributs ***center*** (qui est un **Point**) et un entier ***radius***.

⇨ *Comment savoir si un point p se trouve à l'intérieur du disque ?*



Dans ce dessin, nous voyons que le point ***p*** se trouve sur le disque si la distance entre ce point ***p*** et le centre est plus petite que le radius du disque :

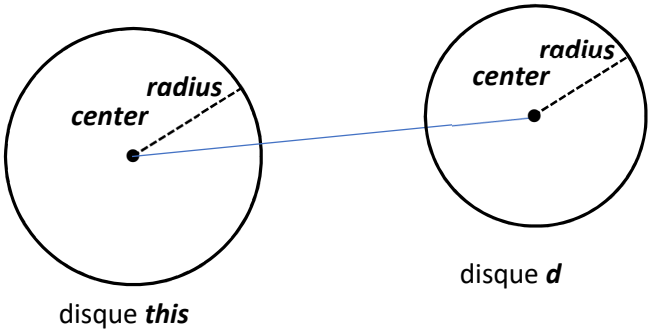
```
public boolean isOnDisk(Point p) {
    return radius>distance(center, p);
}
```

Si le bord compte aussi, il faut écrire >= au lieu de >.

Deux disques se touchent

Supposons une classe **Disk** qui possède les attributs ***center*** (qui est un **Point**) et un entier ***radius***.

⇨ *Comment savoir, pour un disque donné **this**, s'il touche un autre disque **d** ?*



Dans ce dessin, nous voyons que les disques se touchent si la distance entre les deux centres est inférieure à la somme des deux radius. Notons encore que les deux radius de sont pas forcément égaux :

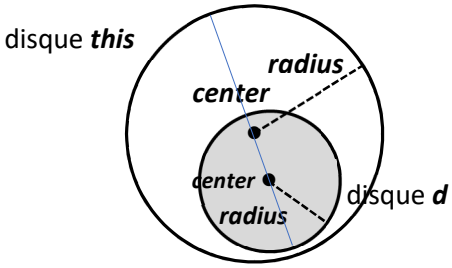
```
public boolean isTouching(Disk d) {
    return radius+d.radius>=distance(center, d.center);
}
```

Deux disques se chevauchent entièrement

Supposons une classe **Disk** qui possède les attributs ***center*** (qui est un **Point**) et un entier ***radius***.

⇨ *Comment savoir, pour un disque donné **this**, s'il couvre entièrement un autre disque **d** ?*

Si le disque ***d*** est dessiné d'abord, alors il sera entièrement caché. S'il est dessiné après, alors il sera dessiné entièrement dans le disque ***this***.



Dans ce dessin, nous voyons que le disque ***d*** se trouve entièrement dans le disque ***this*** si la distance entre les deux centres est inférieure à la différence des deux radius. Là aussi les deux radius de sont pas forcément égaux. :

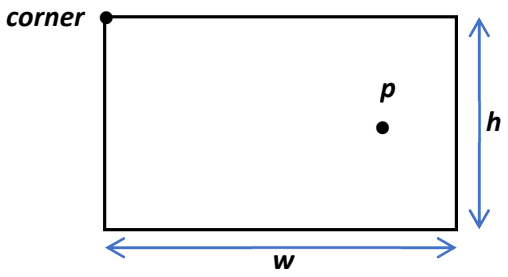
```
public boolean isCovering(Disk d) {
    if (radius<d.radius) return false;
    return radius-d.radius>=distance(center, d.center);
}
```

Notons encore que le disque ***this*** ne peut évidemment pas couvrir le disque ***d*** si le radius de ***this*** est inférieur au radius de ***d***. Cette condition a été rajoutée séparément dans la méthode, non qu'elle soit obligatoire, mais afin de abbréger le calcul.

Un point se trouve sur un rectangle

Supposons une classe **Rect** qui possède les attributs ***corner*** (le point en haut à gauche du rectangle de la classe **Point**) et deux entiers ***w*** et ***h*** qui désignent la largeur et la hauteur du rectangle. Notons que tous les dessins et toutes les méthodes dans la suite devraient être adaptés si la définition d'un rectangle change, p.ex. un Point ***center*** avec deux entiers ***w*** et ***h***, ou deux points opposés ***p1*** et ***p2*** du rectangle.

⇨ *Comment savoir si un point p se trouve à l'intérieur du rectangle ?*



Dans ce dessin, nous voyons que le point p se trouve sur le rectangle si sa coordonnée ***p.x*** se trouve entre le côté gauche et le côté droit et qu'en même temps sa coordonnée ***p.y*** se trouve entre le côté supérieur et le côté inférieur, se qui fait en tout 4 conditions :

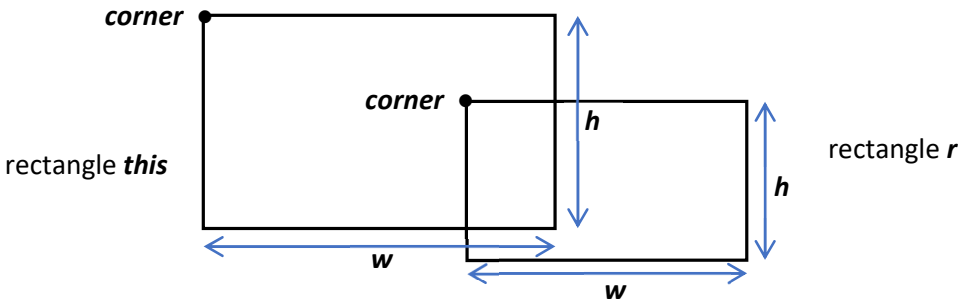
```
public boolean isOnRect(Point p) {
    return p.x>corner.x && p.x<corner.x+w &&
           p.y>corner.y && p.y<corner.y+h);
}
```

Si le bord compte aussi, il faut écrire <= et >= au lieu de < et >.

Deux rectangles se touchent

Supposons une classe **Rect** qui possède les attributs ***corner*** (le point en haut à gauche du rectangle de la classe **Point**) et deux entiers ***w*** et ***h*** qui désignent la largeur et la hauteur du rectangle.

⇨ *Comment savoir, pour un rectangle donné **this**, s'il touche un autre rectangle **r** ?*



Pour résoudre ce problème, posons la question différemment :

⇨ *Quelle sont les conditions, pour qu'un rectangle donné **this** ne touche pas un autre rectangle **r** ?*

Dans ce dessin, nous voyons qu'il suffit qu'une seule des conditions suivantes soit vérifiée :

- le côté droit de ***this*** se trouve à gauche du côté gauche de ***r***
- le côté gauche de ***this*** se trouve à droite du côté droit de ***r***
- le côté supérieur de ***this*** se trouve au-dessus du côté inférieur de ***r***
- le côté inférieur de ***this*** se trouve au-dessous du côté supérieur de ***r***

On peut donc écrire, avec ces mêmes 4 conditions :

```
public boolean isTouching(Rect r) {
    return !(corner.x+w < r.corner.x ||
             corner.x   > r.corner.x+r.corner.w ||
             corner.y+h < r.corner.y ||
             corner.y   > r.corner.y+r.corner.h);
}
```

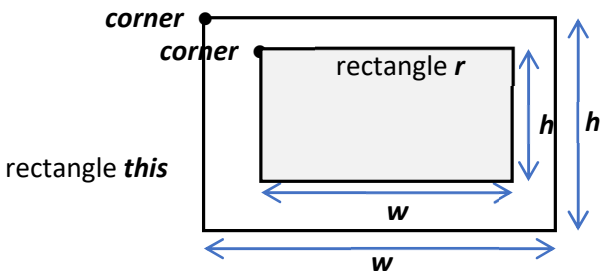
Ou bien, en transformant avec *la loi de Morgan* !(A||B) = (!A)&&(!B) :

```
public boolean isTouching(Rect r) {
    return corner.x+w >= r.corner.x &&
           corner.x   <= r.corner.x+r.corner.w &&
           corner.y+h >= r.corner.y &&
           corner.y   <= r.corner.y+r.corner.h;
}
```

Deux rectangles se chevauchent entièrement

Supposons une classe **Rect** qui possède les attributs ***corner*** (le point en haut à gauche du rectangle de la classe **Point**) et deux entiers ***w*** et ***h*** qui désignent la largeur et la hauteur du rectangle.
Si le rectangle ***r*** est dessiné d'abord, alors il sera entièrement caché. S'il est dessiné après, alors il sera dessiné entièrement dans le rectangle ***this***.

*Comment savoir, pour un rectangle donné **this**, s'il couvre entièrement un autre rectangle **r** ?*



Dans ce dessin, nous voyons que 4 conditions doivent être réunies pour cela :

- le côté gauche de ***this*** se trouve à gauche du côté gauche de ***r***
- le côté droit de ***this*** se trouve à droite du côté droit de ***r***
- le côté supérieur de ***this*** se trouve au-dessus du côté supérieur de ***r***
- le côté inférieur de ***this*** se trouve au-dessous du côté inférieur de ***r***

On peut donc écrire, avec ces mêmes 4 conditions :

```
public boolean isCovering(Rect r) {
    return corner.x <= r.corner.x && corner.x+w >= r.corner.x+r.corner.w &&
           corner.y <= r.corner.y && corner.y+h >= r.corner.y+r.corner.h);
}
```