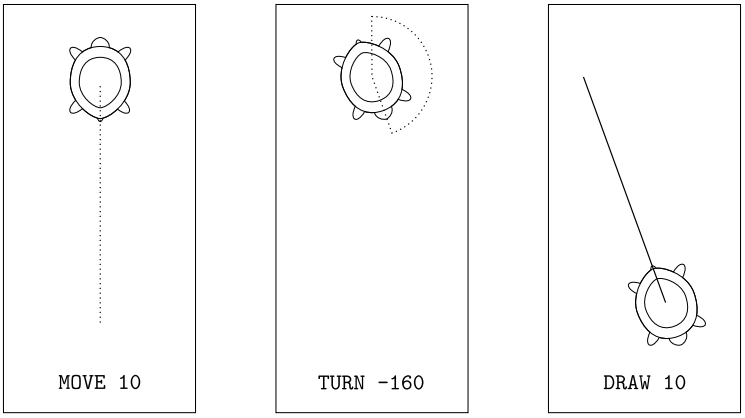
Programmation avancée
TP 7 – Révisions

V. Padovani, PPS - IRIF

En Infographie, une *tortue* est un robot virtuel muni d'un crayon et se déplaçant sur un plan horizontal. La Tortue peut effectuer trois sortes d'actions :

- avancer de n unités de distance (`MOVE`),
- avancer de n unités de distance en traçant un trait (`DRAW`),
- tourner sur place de n degrés (`TURN`).

Le paramètre n de chaque sorte d'action est un entier pouvant être négatif ou positif.



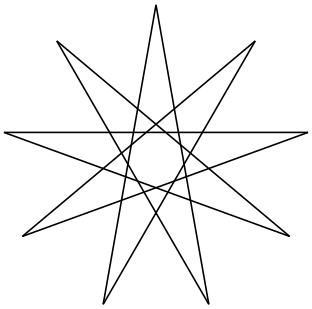
Le langage utilisé pour programmer la tortue est le langage LOGO. On se limitera ici à une version bridée du langage ne contenant que 4 formes d'instructions :

- `MOVE N`
- `DRAW N`
- `TURN N`
- `REPEAT N [`$instr_1$`, ..., `$instr_m$`]`

Le paramètre `N` de chaque forme est une toujours une constante entière. Les trois premières formes correspondent aux trois sortes d'actions ci-dessus. La quatrième permet de répéter `N` fois la suite d'instructions ($instr_1$, ..., $instr_k$). – zéro fois si `N` est négatif. Les `REPEAT` peuvent évidemment s'imbriquer. Ceci est par exemple un programme LOGO :

```
TURN 90
MOVE 160
TURN -90
MOVE 200
REPEAT 9 [DRAW 200, TURN 160]
```

Les quatre premières instructions ne sont destinées qu'à placer et orienter la Tortue, en préparation de la boucle `REPEAT`, qui trace la figure suivante :



L'objectif de ce problème est de trouver le moyen de représenter un programme LOGO en Java sous la forme d'un *tableau d'objets*, et d'écrire le mécanisme permettant de simuler l'exécution de ce programme par une Tortue.

La représentation de la Tortue n'est pas à écrire. Vous ferez simplement l'hypothèse qu'une certaine classe implémente l'interface suivante, l'invocation de chaque méthode effectuant l'action associée de manière naturelle à son nom – peu importe comment :

```
interface Turtle {
  void move (int n);
  void draw (int n);
  void turn (int n);
}
```

Un point important : votre implémentation ne doit contenir *aucun* `if`, en exploitant de manière simple mais appropriée les ressources de la liaison dynamique. Cette consigne est impérative.

1. Une seule hiérarchie de classes suffit. Après l'avoir conçue, donnez la suite d'instructions permettant de construire la représentation exacte du programme donné en exemple ci-dessus.
2. Donnez ensuite l'implémentation complète de chaque classe. La classe au sommet de cette hiérarchie devra contenir une méthode statique permettant d'exécuter la représentation de tout programme par celle de toute Tortue.

En bonus

Si vous avez fini *tout* ce qui précède et s'il vous reste du temps, donner une implémentation concrète de l'interface `Turtle`. Une tortue n'a pas à savoir dans quoi elle inscrit son tracé, une simple instance de `Graphics` lui suffit. Les méthodes suivantes pourront vous être utiles :

- `double Math.toRadiants (double d)`
- `double Math.cos (double d)`,
- `double Math.sin (double d)`
- `void drawLine(int x1, int y1, int x2, int y2)` invocable sur un `Graphics`.