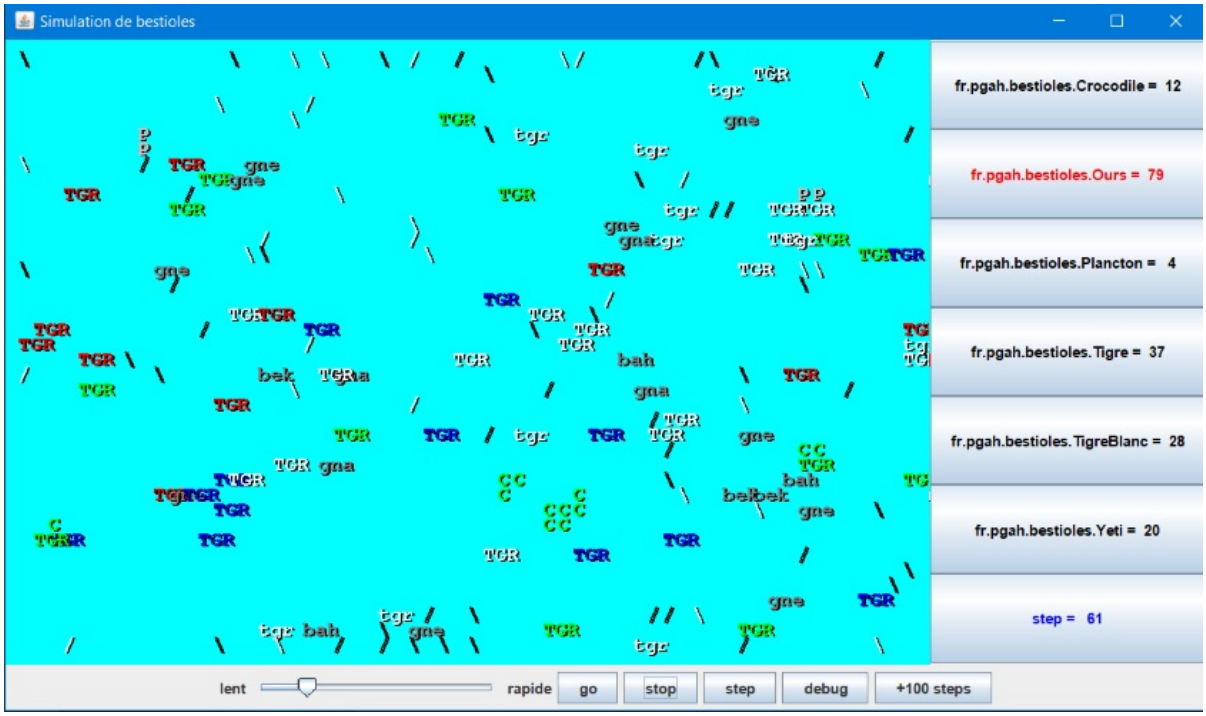
Bestioles

Vous devez définir un ensemble de classes qui définissent le comportement de certains animaux. Vous disposez d'un programme fonctionnel qui fait tourner une simulation d'un monde peuplé d'animaux. Chaque animal se comporte différemment. Voici ce à quoi ça ressemble :



bestioles

Commencez par mettre en place le programme en intégrant les différents fichiers de la simulation dans un répertoire unique sur votre IDE :

- Bestiole.java
- BestioleInfo.java
- BestiolePanel.java
- BestioleModel.java
- BestioleMain.java
- BestioleFrame.java
- Plancton.java
- Crocodile.java

Chaque objet est une « bestiole » ; `Bestiole` est une superclasse qui définit le comportement par défaut. Vous devez écrire cinq classes, chacune représentant un animal diffférent : `Ours`, `Tigre`, `TigreBlanc`, `Yeti`, `ChatNinja`. Ce sont toutes des sous-classes (et donc elles dérivent) de `Bestiole`. À chaque tour de la simulation, on demande à chaque bestiole trois informations :

1. Que vas-tu faire ?
2. De quelle couleur es-tu ?
3. Par quelle chaîne de caractère doit-on te représenter ?

Ces trois informations sont fournies par des méthodes publiques présentes dans la superclasse. Vous devrez les redéfinir et programmer le comportement correct pour chaque bestiole.

```
public Action getAction(BestioleInfo info) {
    ...
}

public Color getCouleur() {
    ...
}

public String toString() {
    ...
}
```

Par exemple, voici une classe `Plancton` dont les objets appaîtraient toujours sous la forme d'un « P » vert et qui essaieraient toujours d'infecter n'importe quelle bestiole qui se trouve devant eux :

```
import java.awt.*;

public class Plancton extends Bestiole {
    public Action getAction(BestioleInfo info) {
        return Action.INFECTER;
    }

    public Color getCouleur() {
        return Color.GREEN;
    }

    public String toString() {
```

```
        return "P";
    }
}
```

Toutes vos classes d'animaux auront ce format. Il sera bien sûr possible d'ajouter des méthodes privées et des attributs privés pour impémenter le comportement de chaque bestiole.

> Note : l'importation de `java.awt` permet d'accéder à la classe `Color`.

Comportement des classes

Ours

- Constructeur : `public Ours(boolean polaire)`
- `getCouleur` :
  - `Color.WHITE` pour un ours polaire (si `polaire` est `true`)
  - `Color.BLACK` sinon (si `polaire` est `false`)
- `toString` : alterne à chaque mouvement entre `/` et `\` (en commençant avec un slash `/`).
- `getAction` :
  - infecte si un ennemi est en face,
  - sinon saute si possible,
  - sinon tourne à gauche.

Tigre

- Constructeur : `public Tigre()`
- `getCouleur` : choisit aléatoirement l'une des trois couleurs : `Color.RED, Color.GREEN, Color.BLUE` et utilise cette couleur pendant trois mouvements, puis choisit de nouveau une couleur aléatoirement pour les trois mouvements suivants, etc.
- `toString` : retourne toujours `"TGR"`
- `getAction` :
  - infecte si un ennemi est en face,
  - sinon si un mur est en face ou à droite, alors tourne à gauche,
  - sinon si un autre tigre est en face, alors tourne à droite,
  - sinon saute.

TigreBlanc

- Constructeur : `public TigreBlanc()`
- `getCouleur` : toujours `Color.WHITE`
- `toString`
  - `"tgr"` s'il n'a pas encore tué d'autres bestioles,
  - `"TGR"` sinon.
- `getAction` : pareil que `Tigre`

Yeti

- Constructeur : `public Yeti()`
- `getCouleur` : toujours `Color.GRAY`
- `toString` :
  - `"gna"` pour les six premiers mouvements,
  - puis `"gne"` pour les six mouvements suivants,
  - puis `"bah"` pour les six mouvements suivants,
  - puis `"bek"` pour les six mouvements suivants,
  - et on répète.
- `getAction` :
  - infecte si un ennemi est en face,
  - sinon saute si possible,
  - sinon tourne à droite.

ChatNinja

- Constructeur : `public ChatNinja()`
- `getCouleur` : vous décidez.
- `toString` : vous décidez.
- `getAction` : Vous décidez.

Les méthodes

getCouleur()

- C’est la plus simple, je vous conseille de commencer par celle-là.
- Cette méthode doit retourner la couleur avec laquelle doit être affichée la bestiole à un moment donné.
- Les couleurs sont représentées par les constantes définies de le package java.awt, par exemple `Color.White`.
- Pour les couleurs aléatoires, chaque possibilité doit avoir une chance égale. Utilisez un objet de type `Random` ou bien la méthode statique `Math.random()`.
- Si la couleur change en fonction des mouvements, il vous faudra un moyen de compter ces mouvements (ce sera un état supplémentaire de l’objet).

toString()

- Doit retourner le texte à afficher pour dessiner la bestiole (parfois un caractère, parfois plusieurs).
- Si ça change en fonction des mouvements, il faudra aussi garder trace des mouvements dans l’état de l’objet.

getAction()

- Retourne l’une de ces quatre actions (définies dans l’énumération `Action`) :
    1. `Action.SAUTER` : avance d’une case dans la direction actuelle.
    2. `Action.GAUCHE` : rotation anti-horaire de 90 degrés.
    3. `Action.DROITE` : rotation horaire de 90 degrés.
    4. `Action.INFECTER` : tente d’infecter la bestiole en face, c’est-à-dire de la transformer en une bestiole de sa propre espèce.
- Vous aurez besoin de l’information sur la zone proche pour déterminer quelle action entreprendre. Par exemple : ours, tigres, tigres blancs et chats ninjas devraient tous INFECTER si un ennemi est en face d’eux.
- Vous pouvez connaître ce qui est autour de la bestiole en interrogeant l’objet `BestioleInfo` passé en paramètre de `getAction` :
    - `public Voisin getEnFace()` : retourne le voisin en face (valeurs possibles : `Voisin.MUR, Voisin.RIEN, Voisin.MEME, Voisin.AUTRE`).
    - `public Voisin getDerriere()` : retourne le voisin derrière.
    - `public Voisin getAGauche()` : retourne le voisin à gauche.
    - `public Voisin getADroite()` : retourne le voisin à droite.

- `public Direction getDirectionActuelle()` : retourne la direction actuelle de la bestiole (valeurs possibles : `Direction.NORD, Direction.SUD, Direction.EST, Direction.Ouest`)
- `public boolean menaceEnFace()` : y a-t-il un ennemi en face qui vous fait regarde ?
- `public boolean menaceDerriere()` : y a-t-il un ennemi derrière qui vous regarde ?
- `public boolean menaceAGauche()` : y a-t-il un ennemi sur la gauche qui vous regarde ?
- `public boolean menaceADroite()` : y a-t-il un ennemi sur la droite qui vous regarde ?

FAQ

- Vous n'écrirez pas de méthode `main` ; votre code ne contrôlera pas la boucle principale ni l'interface graphique.
- Vous définissez une série d'objets qui vont prendre part à un système plus large.
- Par exemple, vous ne pourrez pas faire en sorte qu'une bestiole fasse plusieurs mouvements d'un coup. La seule façon qu'a une bestiole de bouger, c'est d'attendre que le simulateur lui demande en appelant `getAction` sur elle. Le simulateur contrôle, pas vos bestioles.
- Les bestioles bougent dans un monde fini, délimité par les quatre côtés de la zone de l'écosystème.
- Si une classe utilise toujours les mêmes Strings ou couleurs, vous devriez les déclarer en tant que constantes.
- Le simulateur est composé de plusieurs classes support (`BestioleModel`, `BestioleFrame` …) ; vous pouvez les ignorer ou les étudier pour comprendre comment ça fonctionne.
- Vous modifierez `BestioleMain` pour ajouter des bestioles à la simulation quand les classes seront prêtes (en décommentant les lignes de code qui ajoutent les bestioles à l'écosystème).

Tests

- Le simulateur montre bien où sont les bestioles, mais on ne voit pas aisément quelle est leur direction actuelle. Quand on clique sur le bouton “debug”, les bestioles sont affichées avec des flèches qui indiquent leur direction.
- Le simulateur indique aussi le nombre de “pas” (de “steps”, d’étapes, d’itérations) courants de la simulation.
- Voici quelques suggestions pour tester vos bestioles :
    - **Ours** : mettez juste 30 ours dans la simulation. À peu près la moitié devraient être blancs, et l’autre moitié noirs. Ils devraient tous commencer à s’afficher avec des `/`, et alterner avec des `\`. Utilisez le bouton “step” pour vérifier. Les ours auront tendance à aller vers les murs et à les longer dans le sens anti-horaire. Ils vont de temps en temps se rentrer dedans et changer de direction.
    - **Tigres** : mettez juste 30 tigres dans le simulateur. Vous devriez avoir une dizaine de rouges environ, une dizaine de verts et une dizaine de bleus. Utilisez le bouton “step” pour vérifier que les couleurs changent tous les trois mouvements. Quand ils se cognent contre un mur, vous devriez les voir faire demi-tour. Ils se rentreront également dedans parfois, et ils changeront de direction. Vous ne devriez pas les voir s’agglutiner. Les tigres blancs se comporteront de la même manière mais ils seront blancs. Ils seront aussi en lettres minuscules tant qu’ils n’auront pas infecté une autre bestiole, après quoi ils seront en majuscules.
    - **Yeti** : Essayez 30 yétis. Vous devez observer la séquence `gna/gne/bah/bek`, sur 6 x 4 = 24 mouvements, puis retour à `gna` à partir du step 24 (ça commence à 0) pour une nouvelle boucle. Les yétis ont la même tendance que les ours à se diriger vers les murs, mais eux vont tourner dans le sens horaire une fois qu’il les ont atteints.