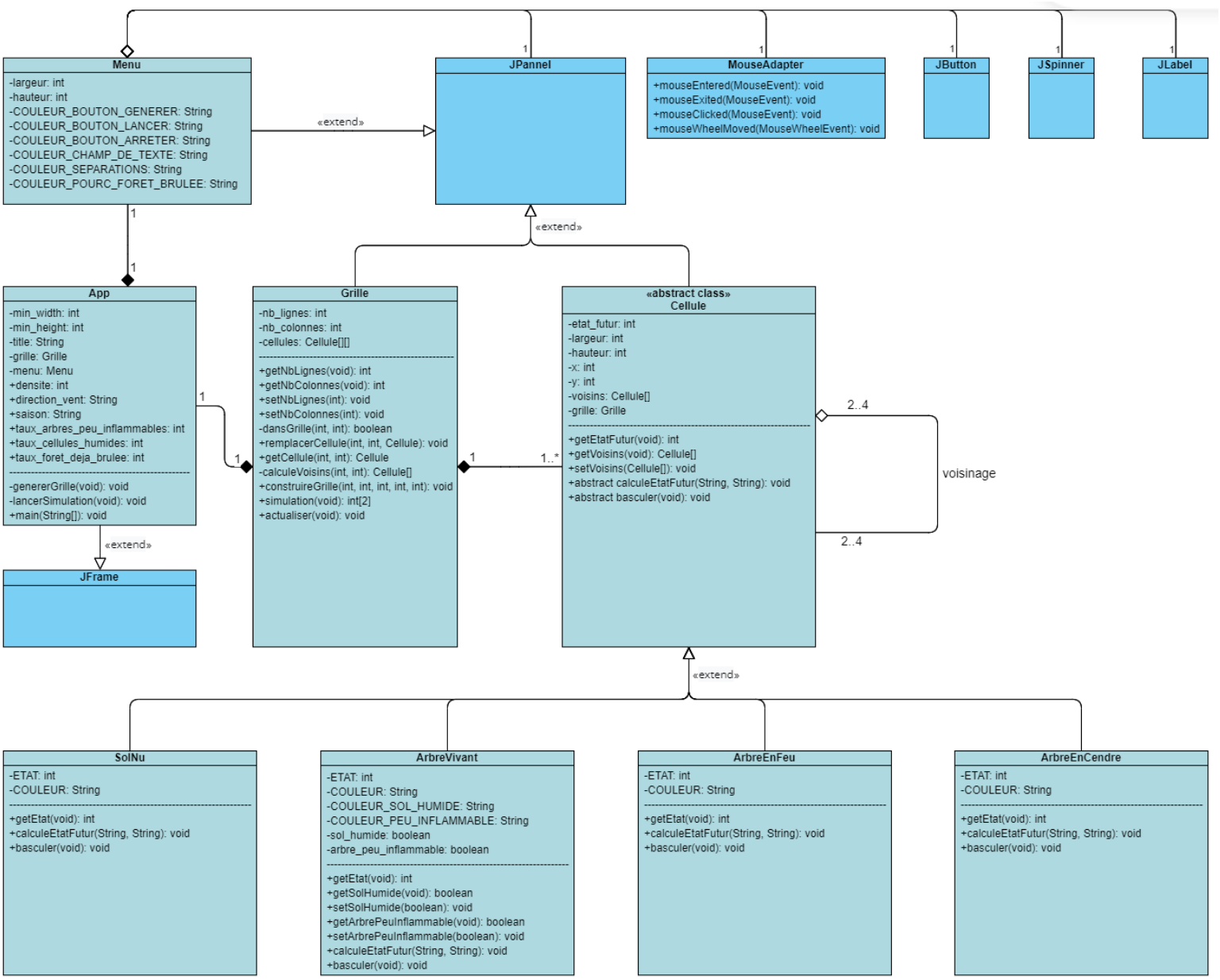
DESCRIPTION DIAGRAMME UML



Le projet comprendra les classes suivantes :

- La classe **App** : Fenêtre principale du projet, point de départ de l'application
- La classe **Menu** : Contient les boutons et les champs qui permettent de configurer la Grille.
- La classe **Grille** : représente la forêt
- La classe **Cellule** : représente les composantes de la foret (sol nu, arbre vivant, arbre en feu, arbre en cendres)
- Les classes **SolNu**, **ArbreVivant**, **ArbreEnFeu** et **ArbreEnCendre :** qui héritent de la classe Cellule et qui permettent de modéliser les différentes variétés de cellules.

1. La classe **App**

Cette classe est une extension (« extend ») de la classe JFrame de Java.

Attributs :

- *int min_width* : largeur minimale de la fenêtre

- *int min_height* : hauteur minimale de la fenêtre
- *String title* : titre de la fenêtre
- *Grille grille* : c'est la grille qui modélise la forêt
- *Menu menu* : c'est le panneau contenant le menu de l'application
- *int densité* : densité des arbre dans la forêt en pourcentage
- *String direction_vent* : indique la direction du vent (NORD, SUD, EST, OUEST)
- *String Saison* : HIVER, PRINTEMPS, ETE, AUTOMNE
- *int taux_arbres_peu_inflammable* : taux en pourcentage d'arbre à faible inflammabilité
- int *taux_cellules_humides* : taux en pourcentage de cellules ayant un sol humide
- *int taux_foret_deja_brulee* : taux en pourcentage de forêt déjà brulée

Méthodes :

- *App App (int minWidth, int minHeight, String title)* : Constructeur de la classe. C'est ici que la fenêtre de l'application sera créée, ainsi que tous les différents composants graphiques.
- *void genererGrille ()* : Méthode qui permet d'initialiser de façon aléatoire la grille avec un nombre de colonnes et de lignes, une densité, un pourcentage de cellules humides et un pourcentage d'arbres peu inflammable fournis par l'utilisateur. Cette méthode est appelée après un click de l'utilisateur sur le bouton **Générer**.
- *void lancerSimulation ()* : Méthode qui utilise les règles de propagation du feu pour mettre la grille à jour de façon itérative jusqu'à ce qu'il n'y ai plus de changement dans la grille ou que l'utilisateur clique sur le bouton **Arrêter**.
- *public static void main(String[] args)* : C'est la méthode main, point d'entrer de l'application qui fera appel au constructeur de la classe App pour lancer l'application.

2. La classe **Menu**

Cette classe est une extension de JPannel. Elle contient tous les composants graphiques (Boutons, etc.) permettant de configurer la grille et de lancer la simulation.

Attributs :

- *int largeur* : largeur minimale du panneau
- *int hauteur* : hauteur minimale du panneau
- *final static String COULEUR_BOUTON_GENERER*: couleur en hexadecimal du bouton « Générer ».
- *final static String COULEUR_BOUTON_LANCER*: couleur en hexadecimal du bouton « Lancer ».
- *final static String COULEUR_BOUTON_ARRETER*: couleur en hexadecimal du bouton « Arrêter ».
- *final static String COULEUR_CHAMP_DE_TEXTE*: couleur en hexadecimal des champs de texte qui permettent à l'utilisateur de faire des choix (JSpinner).
- *final static String COULEUR_SEPARATIONS*: couleur en hexadecimal des lignes de séparation.
- *final static String COULEUR_POURC_FORET_BRULEE*: couleur en hexadecimal d'affichage du pourcentage de la forêt brulée.

3. La classe **Grille**

Cette classe est une extension de la classe JPannel de Java. Elle permet d'implémenter la grille sur laquelle se déroulera la simulation.

Attributs :

- *int nb_lignes* : nombre de lignes de la grille

- *int nb_colonnes* : nombre de colonnes de la grille
- *Cellule[][] cellules* : tableau à deux dimensions contenant toutes les cellules de la grille

Méthodes :

- *Grille Grille (nbLignes, nbColonnes)* : constructeur de la classe
- *int getNbLignes ()* : méthode qui renvoie le nombre de lignes de la grille
- *int getNbColonnes ()* : méthode qui renvoie le nombre de colonnes de la grille
- *int setNbLignes (int nbLignes)* : méthode qui permet de modifier le nombre de lignes de la grille
- *int setNbColonnes (int nbColonnes)* : méthode qui permet de modifier le nombre de colonnes de la grille.
- *boolean dansGrille (int i, int j)* : méthode qui permet de vérifier si la cellule de coordonnées (i, j) fait partie la grille. Elle retourne True si le cellule (i, j) est dans la grille et False sinon
- *void remplacerCellule (int i, int j, Cellule cellule)* : méthode qui permet de remplacer la cellule de la case (i, j) de la grille par une nouvelle. Le paramètre "cellule" représente la cellule à utiliser pour remplacer celle de la case (i, j) de la grille.
- *Cellule getCellule (int i, int j)* : Méthode qui renvoie la cellule située dans la case (i, j) de la grille.
- *Cellule calculeVoisins (int i, int j)* : renvoie la liste des voisins de la cellule (i, j) ; [vg, vh, vd, vb] ; si un voisin manque le remplacer par « null » .
- *construireGrille (int nbLignes, int nbColonnes, int densité, int tauxCellulesHumides, int tauxArbresPeuInflammables)* : méthode qui permet de créer aléatoirement la Grille avec les paramètres saisies par l'utilisateur. Cette méthode intervient lors de l'instanciation de la grille et lorque l'utilisateur clique sur le bouton **Générer**.
- *int[2] simulation ()* : cette méthode passe en revue toutes les cellules de la grille et calcule leur état futur. Elle retourne deux entiers ; le premier contiendra *0* si on doit arrêter la simulation (il n'y a plus d'arbres en feu) et *1* sinon ; le second contiendra le pourcentage de la forêt déjà brulée.
- *void actualiser (void)* : permet de basculer toutes les cellules de la grille dans leur état futur.

4. La classe abstraite **Cellule**

Cette classe abstraite est une extension de la classe JPannel de Java. Elle permet de modéliser les composantes de la forêt

Attributs :

- *int etat_futur* : représente l'état futur de la cellule à la prochaine itération.
- *int largeur* : représente la largeur de la cellule.
- *int hauteur* : représente la hauteur de la cellule.
- *int x* : numero de ligne ou se trouve la cellule dans la grille
- *int y* : numero de colonne ou se trouve la cellule dans la grille
- *Cellule[] voisins* : représente la liste des voisins de la cellule dans l'ordre suivant : [voisin gauche, voisin du haut, voisin de droite, voisin du bas]
- *Grille grille* : représente la grille dans laquelle se trouve la cellule

Méthodes :

- *Cellule Cellule (int largeur, int hauteur, int x, int y, Grille grille)* : constructeur de la classe
- *Cellule[] getVoisins ()* : Retourne la liste des voisins de la cellule dans l'ordre suivant : [voisin gauche, voisin du haut, voisin de droite, voisin du bas]

- *void setVoisins (Cellule[] voisins)* : Permet d'affecter des voisins à la cellule.
- abstract void calculeEtatFutur (string direction_vent, string saison) : méthode abstraite très important pour le projet. Définit par les classes concrètes qui hérite de Cellule, elle permet de calculer l'état futur d'une cellule en se basant sur plusieurs critères à savoir, le voisinage, la direction du vent, la saison, l'humidité du sol et le type de végétation. A la fin de ce document vous trouverez un pseudo code pour cette méthode.
- *abstract void basculer ()* : méthode qui crée une nouvelle cellule de type etat_futur et l'utilise pour remplacer la cellule courante dans la grille.

5. La classe abstraite **SolNu**

Cette classe est une extension de la classe Cellule. Elle permet de modéliser le sol nu.

Attributs :

- *final static int ETAT* : constante qui représente l'état de la cellule. Sa valeur est 0.
- *final static int COULEUR* : constante qui représente la couleur de la cellule.

Méthodes :

- *int getEtat ()* : méthode qui renvoie l'état de la cellule, soit 0.
- void calculeEtatFutur (string direction_vent, string saison) : retourne ETAT, car la terre nue ne change pas d'état.
- *void basculer ()* : ne fait rien car état futur égal à a état courant.

6. La classe abstraite **ArbreVivant**

Cette classe est une extension de la classe Cellule. Elle permet de modéliser un arbre vivant.

Attributs :

- *final static int ETAT* : constante qui représente l'état de la cellule. Sa valeur est 1.
- *final static int COULEUR* : constante qui représente la couleur de la cellule.
- *final static int COULEUR_SOL_HUMIDE* : constante qui représente la couleur de l'arbre lorsque celui-ci est placé sur un sol humide.
- *final static int COULEUR_PEU_INFLAMMABLE* : constante qui représente la couleur des bordures de la cellule lorsque celle-ci contient un arbre peu inflammable.
- *boolean sol_humide* : indique si le sol de la cellule est humide ou pas.
- *boolean arbre_peu_inflammable* : indique si l'arbre est peu inflammable ou pas.

Méthodes :

- *int getEtat ()* : méthode qui renvoie l'état courant de la cellule, soit 1.
- *boolean getSolHumide ()* : getter de l'attribut *sol_humide*.
- *void setSolHumide (boolean sol_humide)* : setter de l'attribut *sol_humide*.
- *boolean getArbrePeuInflammable ()* : getter de l'attribut *arbre_peu_inflammable.*
- *void setArbrePeuIunflammable (boolean arbre_peu_inflammable)* : getter de l'attribut *arbre_peu_inflammable.*
- void calculeEtatFutur (string direction_vent, string saison) : permet de calculer l'etat futur de la cellule en se basant sur plusieurs critères à savoir, le voisinage, la direction du vent, la saison, l'humidité du sol et le type de végétation. Voir le pseudo code à la fin du document pour mieux comprendre.

- *void basculer ()* : Si l'état futur reste arbre vivant (1), alors la méthode ne fait rien. Sinon elle crée une nouvelle cellule de type etat_futur (arbre en feu, 2) et l'utilise pour remplacer la cellule courante dans la grille.

7. La classe abstraite **ArbreEnFeu**

Cette classe est une extension de la classe Cellule. Elle permet de modéliser un arbre en feu.

Attributs :

- *final static int ETAT* : constante qui représente l'état de la cellule. Sa valeur est 2.
- *final static int COULEUR* : constante qui représente la couleur de la cellule.

Méthodes :

- *int getEtat ()* : méthode qui renvoie l'état de la cellule, soit 2.
- void calculeEtatFutur (string direction_vent, string saison) : retourne 3 (code de l'état d'un arbre en cendre), car tout arbre en feu devient un arbre en cendre.
- *void basculer ()* : méthode qui crée une nouvelle cellule de type arbre en cendre (3) et l'utilise pour remplacer la cellule courante dans la grille.

8. La classe abstraite **ArbreEnCendre**

Cette classe est une extension de la classe Cellule. Elle permet de modéliser un arbre en cendre.

Attributs :

- *final static int ETAT* : constante qui représente l'état de la cellule. Sa valeur est 3.
- *final static int COULEUR* : constante qui représente la couleur de la cellule.

Méthodes :

- *int getEtat ()* : méthode qui renvoie l'état de la cellule, soit 3.
- void calculeEtatFutur (string direction_vent, string saison) : retourne ETAT, car un arbre en cendre ne change pas d'état.
- *void basculer ()* : ne fait rien car état futur égal à a état courant.

Pseudo code général pour la méthode calculeEtatFutur

```
import java.util.Random;
void calculeEtatFutur(String direction_vent, String saison) {
   si je suis un sol nu (etat = 0) ou un arbre en cendre (etat = 3) alors etat_futur = etat finsi
   si je suis un arbre en feu (etat = 2) alors etat_futur = 3 (arbre en cendre) finsi
   si je suis un arbre vivant et aucun de mes voisins n'est en feu alors je reste vivant (etat_futur = etat) finsi
   si je suis un arbre vivant et au moins un de mes voisin est en feu alors je calcule p (probabilité de passer en feu)
      /* Impact du voisinage */
      si j'ai n voisins en feu alors p = 0.8+(n * 0.05) finsi
      /* Impact de la direction du vent */
      feu_vers_cellule = FALSE
      si le voisin de gauche est en feu et le vent va vers l'EST alors feu_vers_cellule = TRUE finsi
      si le voisin de droite est en feu et le vent va vers l'OUEST alors feu_vers_cellule = TRUE finsi
      si le voisin du haut est en feu et le vent va vers le SUD alors feu_vers_cell e = TRUE finsi
      si le voisin du bas est en feu et le vent va vers le NORD alors feu_vers_cellule = TRUE finsi
      si feu_vers_cellule = FALSE et direction_vent <> null alors p = min(0, p - 0.1) finsi
      si feu_vers_cellule = TRUE alors p = max(1, p + 0.5) finsi
```

```
        /* Impact de la saison : l'hiver diminue la probabilité pour un arbre de bruler de 10%, l'été l'augmente
        de 10%, Le printemps et l'automne n'ont aucun effet */
        si saison = 'HIVER' alors p = min(0, p - 0.1) finsi
        si saison = 'ETE' alors p = max(1, p + 0.1) finsi
        /* Impact de l'humidité du sol */
        si sol_humide = TRUE alors p = min(0, p - 0.1) finsi
        /* Impact du type de végétation */
        si arbre_peu_imflamable = TRUE alors p = min(0, p - 0.1) finsi

        /* Calcul de l'état futur à l'aide de la probabilité p
        Random random = new Random();
        double randomValue = random.nextDouble();
        si (randomValue <= 1-p) alors
            etat_futur = etat;
        sinon
            etat_futur = 2;
        finsi
    finsi
}
```