Projet LU2IN002 - 2022-2023

Numéro du groupe de TD/TME : 11 (DM-IM)

Nom : SOUAIBY	*Nom :* SAMAHA
Prénom : Christina	*Prénom :* Elio
N° étudiant : 21102782	*N° étudiant :* 21105733

Thème choisi (en 2 lignes max.)
Le thème du projet est la récolte de raisins dans un Vignoble et leur transformation en Vin, puis la vente de la production.

Schéma UML des classes vision fournisseur (dessin "à la main" scanné ou photo acceptés)
Dernière page

Checklist des contraintes prises en compte:	*Nom(s) des classe(s) correspondante(s)*
Classe contenant un tableau ou une ArrayList	Agent, Simulation
Classe avec membres et méthodes statiques	Agent, MachineAVendanger, Vigne, Simulation
Classe abstraite et méthode abstraite	Vigne, Vin , Agent (méthodes : recolter, sortirTerrain)
Interface	Evolution
Classe avec un constructeur par copie ou clone()	MachineAVendanger
Définition de classe étendant Exception	ListeVideException
Gestion des exceptions	Simulation, TestSimulation

Présentation brève de votre projet (max. 10 lignes) : texte libre expliquant en quoi consiste votre projet.
Notre projet représente une récolte dans un vignoble dans lequel on retrouve 2 types de vignes (Chardonnay et Muscat). La récolte se fait par les hommes (attention, certains sont malhonnêtes !) ou par des Machines à Vendanger (plus efficaces, ne se fatiguent pas contrairement aux hommes qui ont une durée maximale de travail). Pour optimiser l'efficacité de la récolte et éviter la saturation, on gère les entrées et sorties sur le terrain grâce à un attribut d'activité (booléen) pour chaque agent ainsi qu'un compteur d'agents actifs qui ne doit pas dépasser un maximum bien défini. Par respect pour les travailleurs, on s'occupe surtout de sortir du terrain (passer en mode inactif) un homme qui a travaillé pendant sa durée maximale de travail. La variation de la quantité de raisins sur les vignes dépend : - des récoltes faites par les agents et de l'honnêteté des hommes (un voleur cueille plus que ce qu'il doit cueillir et s'enfuit avec les ressources, ce qui affecte aussi la quantité de vin produite) - du passage des saisons (les vignes donnent plus de fruit au fil du temps, mais elles en perdent pendant l'hiver) La quantité de vin produite est vendue (par bouteilles) à chaque fin de journée. Le nombre de ventes est aléatoire car il dépend du nombre de clients du jour.

Copier / coller vos classes et interfaces à partir d'ici :

Signatures

// Agents
public abstract class Agent ;
public class Homme extends Agent ;
public class MachineAVendanger extends Agent ;

// Ressources
public interface Evolution ;
public abstract class Vigne extends Ressource implements Evolution ;
public class VigneChardonnay extends Vigne;
public class VigneMuscat extends Vigne;

public abstract class Vin extends Ressource ;
public class VinChardonnay extends Vin ;
public class VinMuscat extends Vin ;

// Simulation, tests, exceptions
public class ListeVideException extends Exception ;
public class Simulation ;
public class TestSimulation ;

----------AGENT----------

```
import java.util.ArrayList;

public abstract class Agent {
  // Attributs
  protected int x, y;
  protected boolean estActif;
  public final int NBACTIFSMAX = 12;
  protected static int nbActifs = 0;
  protected static int quantiteRecoltes = 0;
  protected static ArrayList<Agent> listeActifs = new ArrayList<>();

  // Constructeurs
  public Agent(int x, int y, Terrain t) {
    this.x = x;
    this.y = y;
    estActif = (t.sontValides(x, y) && (inListeActifs(x,y)==null) && nbActifs < NBACTIFSMAX) ;
    if (estActif) listeActifs.add(this);
    nbActifs = (estActif) ? (nbActifs + 1) : (nbActifs);
  }

  public Agent(Terrain t) {
    this((int) (Math.random() * t.nbLignes), (int) (Math.random() * t.nbColonnes), t);
  }

  // Methodes
  public double distance(int x, int y) {
    return Math.sqrt(Math.pow(x - this.x, 2) + Math.pow(y - this.y, 2));
  }

public void seDeplacer(int xnew, int ynew) {
    if (inListeActifs(xnew,ynew)==null){
      String agent=(this instanceof Homme)?(((Homme)this).getNom()):("Machine "+(((MachineAVendanger)this).getId()));
      System.out.println(agent+" se déplace en ("+xnew+","+ynew+") car la ressource en ("+x+","+y+") est épuisée.");
      x = xnew;
```

```
      y = ynew;
    }
  }

  public void trouverPlusProcheRessource(Terrain t){
    double dmin = 100;
    int a = x;
    int b = y;
    for (int i=0; i<t.nbLignes; i++){
      for (int j=0; j<t.nbColonnes; j++){
        if (t.caseEstVide(i,j) || (i==x && j==y) || (inListeActifs(x, y)!=null && inListeActifs(x, y)!=this))
          continue;
        else{
          if (distance(i,j)<dmin){
            dmin=distance(i,j);
            a=i;
            b=j;
          }
        }
      }
    }
    seDeplacer(a,b);
  }

public Agent inListeActifs(int x, int y){
    if (listeActifs!=null){
      for (Agent a : listeActifs){
        if ((x == a.x) && (y == a.y) && a!=this){
          return a;
        }
      }
    }
    return null;
  }

  public String toString() {
    return " | Position (" + x + "," + y + ") | " + ((estActif) ? "Actif" : "Inactif");
  }

  public abstract void recolter(Ressource r, Terrain t);

  public void transformer(Vin r){
    int bouteille = 200 * (quantiteRecoltes/125);
    quantiteRecoltes = quantiteRecoltes%125;
    r.setQuantite(r.getQuantite() + bouteille);
  }

  public void entrerTerrain(){
    if (!estActif){
      if (nbActifs < NBACTIFSMAX && (inListeActifs(x , y)==null) ) {
        estActif = true;
        if (this instanceof Homme) {
          Homme temp = (Homme)this;
          System.out.println(temp.getNom()+" est rentré(e) sur le terrain");
        }
        if (this instanceof MachineAVendanger) System.out.println("Machine "+((MachineAVendanger)this).getId() + " est rentré(e) sur
le terrain");
        listeActifs.add(this);
        nbActifs++;
      }
    }
  }

  public abstract void sortirTerrain();

  // Accesseurs
  public static int getNbActif() {
    return nbActifs;
  }
  public int getX() {
```

```
    return x;
  }
  public int getY() {
    return y;
  }
  public boolean getEstActif() {
    return estActif;
  }
  public static ArrayList<Agent> getListeActifs(){
    return listeActifs;
  }
}

----------------------------------------------------------------------------HOMME----------------------------------------------------------------------------
public class Homme extends Agent {
  // Attributs
  public final int DUREEMAX = 8;
  private String nom;
  private boolean estVoleur;
  protected int dureeTravail;

  // Constructeurs
  public Homme(int x, int y, String nom, boolean estVoleur, Terrain t) {
    super(x, y, t);
    this.estVoleur = estVoleur;
    this.nom = nom;
    dureeTravail = 0;
  }
  public Homme(int x, int y, String nom, Terrain t) {
    super(x, y, t);
    this.estVoleur = Math.random() < 0.4;
    this.nom = nom;
    dureeTravail = 0;
  }

  public Homme(String nom, Terrain t) {
    super(t);
    this.estVoleur = Math.random() < 0.4;
    this.nom = nom;
    dureeTravail = 0;
  }

  // Methodes
  public void updateDureeTravail() {
    dureeTravail+=2;
  }

  public void sortirTerrain() {
    if (estActif){
      if (dureeTravail >= DUREEMAX || nbActifs >= NBACTIFSMAX || (inListeActifs(x, y)!=null && inListeActifs(x, y)!=this)  ) {
        nbActifs--;
        estActif = false;
        listeActifs.remove(this);
        if (nbActifs >= NBACTIFSMAX) System.out.println("Trop d'agents sur le terrain.");
        if (inListeActifs(x, y)!=null && inListeActifs(x, y)!=this) System.out.println("Il y'a déjà un agent à la place de "+nom+".");
        if (dureeTravail >= DUREEMAX) System.out.println(nom+" a travaillé jusqu'à présent "+dureeTravail+" heures. Il/Elle est
fatigué(e).");
        System.out.println(nom+" est sorti(e) du terrain");
        dureeTravail=0;
      }
    }
  }
  public void recolter(Ressource r, Terrain t) {
    if (r == null) return;
    if (estVoleur) {
      System.out.println("AU VOLEUR ! La récolte en (" + r.getX() + "," + r.getY()
          + ") a été volée par un Homme malhonnête, 50% a été perdu. Le coupable est : " + nom );
      r.setQuantite((int) (0.5 * r.getQuantite()-1));
    } else {
```

```
      System.out.println(nom+ " a récolté "+ (int) (0.2 * r.getQuantite()+1) +" kg de raisins "+ ( (r instanceof
VigneChardonnay)?"Chardonnay":"Muscat")) ;
      quantiteRecoltes += (int) (0.2 * r.getQuantite()+1);
      r.setQuantite((int) (0.8 * r.getQuantite()-1));
    }
    if (r.getQuantite()<=0) r=t.videCase(x,y);
  }
  public String toString() {
    return "Je m'appelle " + nom + ", j'ai travaillé jusqu'à présent " + dureeTravail + " heures de " + DUREEMAX + " heures." +
super.toString();
  }

  // Accesseurs
  public boolean getEstVoleur() {
    return estVoleur;
  }
  public String getNom(){
    return nom;
  }
}

----------------------------------------------------------------------MACHINE A VENDANGER----------------------------------------------------------------
public class MachineAVendanger extends Agent{
  // Attributs
  private static int count = 0;
  private int id;

  // Constructeurs
  public MachineAVendanger(int x, int y, Terrain t){
    super(x,y,t);
    count++;
    id=count;
  }
  public MachineAVendanger(Terrain t){
    super(t);
    count++;
    id=count;
  }
  public MachineAVendanger(MachineAVendanger m, Terrain t){//copie
    this(m.x,m.y, t);
    count++;
    id=count;
  }

// Methodes
public void sortirTerrain(){
    if (estActif){
      if (nbActifs>=NBACTIFSMAX || (inListeActifs(x, y)!=null && inListeActifs(x, y)!=this) ){
        estActif=false;
        nbActifs--;
        listeActifs.remove(this);
        if (nbActifs >= NBACTIFSMAX) System.out.println("Trop d'agents sur le terrain.");
        if (inListeActifs(x, y)!=null && inListeActifs(x, y)!=this) System.out.println("Il y'a déjà un agent à la place de Machine
"+id+".");
        System.out.println("Machine "+id+" est sorti(e) du terrain");
      }
    }
  }
  public String toString(){
    return "Moi machine "+ id + ". Moi pas fatigué."+super.toString();
  }
  public void recolter(Ressource r, Terrain t){
    if (r == null) return;
    quantiteRecoltes+=(int)(0.3*r.getQuantite()+1);
    r.setQuantite((int)(0.7*r.getQuantite()-1));
    System.out.println("Machine "+id+ " a récolté "+ (int) (0.2 * r.getQuantite()+1) +" kg de raisins "+ ( (r instanceof
VigneChardonnay)?"Chardonnay":"Muscat") ) ;
    if (r.getQuantite()<=0) r=t.videCase(x,y);
  }
  public int getId(){
```

```
    return id;
  }
}

----------------------------------------------------------------------------EVOLUTION----------------------------------------------------------------------------
public interface Evolution{
  // La quantité de raisins sur les vignes augmente au fil des saisons, mais diminue en hiver
  public void croissance();
}

----------------------------------------------------------------------------VIGNE----------------------------------------------------------------------------
public abstract class Vigne extends Ressource implements Evolution{
  // Attributs
  private static String[] saisons= {"Printemps", "Ete", "Automne", "Hiver"};
  protected static int saison=0; // 0 : printemps

  // Constructeurs
  public Vigne(String nom){
    super(nom,200);
  }

  // Methodes
  public static void passageSaisons(){
    saison=(saison+1)%4;
  }

  public void croissance(){
   if (saison==3){
      this.setQuantite(this.getQuantite()-25);
    }
    else{
       this.setQuantite(this.getQuantite()+10);
    }
  }

  // Accesseurs
  public static int getSaison(){
    return saison;
  }
  public static String[] get4Saisons(){
    return saisons;
  }
}

----------------------------------------------------------------------------VIGNECHARDONNAY----------------------------------------------------------------------------
public class VigneChardonnay extends Vigne{
  // Constructeurs
  public VigneChardonnay(){
    super("Vigne Chardonnay");
  }
}

----------------------------------------------------------------------------VIGNEMUSCAT----------------------------------------------------------------------------
public class VigneMuscat extends Vigne{
  // Constructeurs
  public VigneMuscat(){
    super("Vigne Muscat");
  }
}

----------------------------------------------------------------------------VIN----------------------------------------------------------------------------
public abstract class Vin extends Ressource{
  // Constructeurs
  public Vin(String nom){
    super(nom,0);
  }

  // Methodes
  public void vendre(){
    if (this.getQuantite()>0){
```

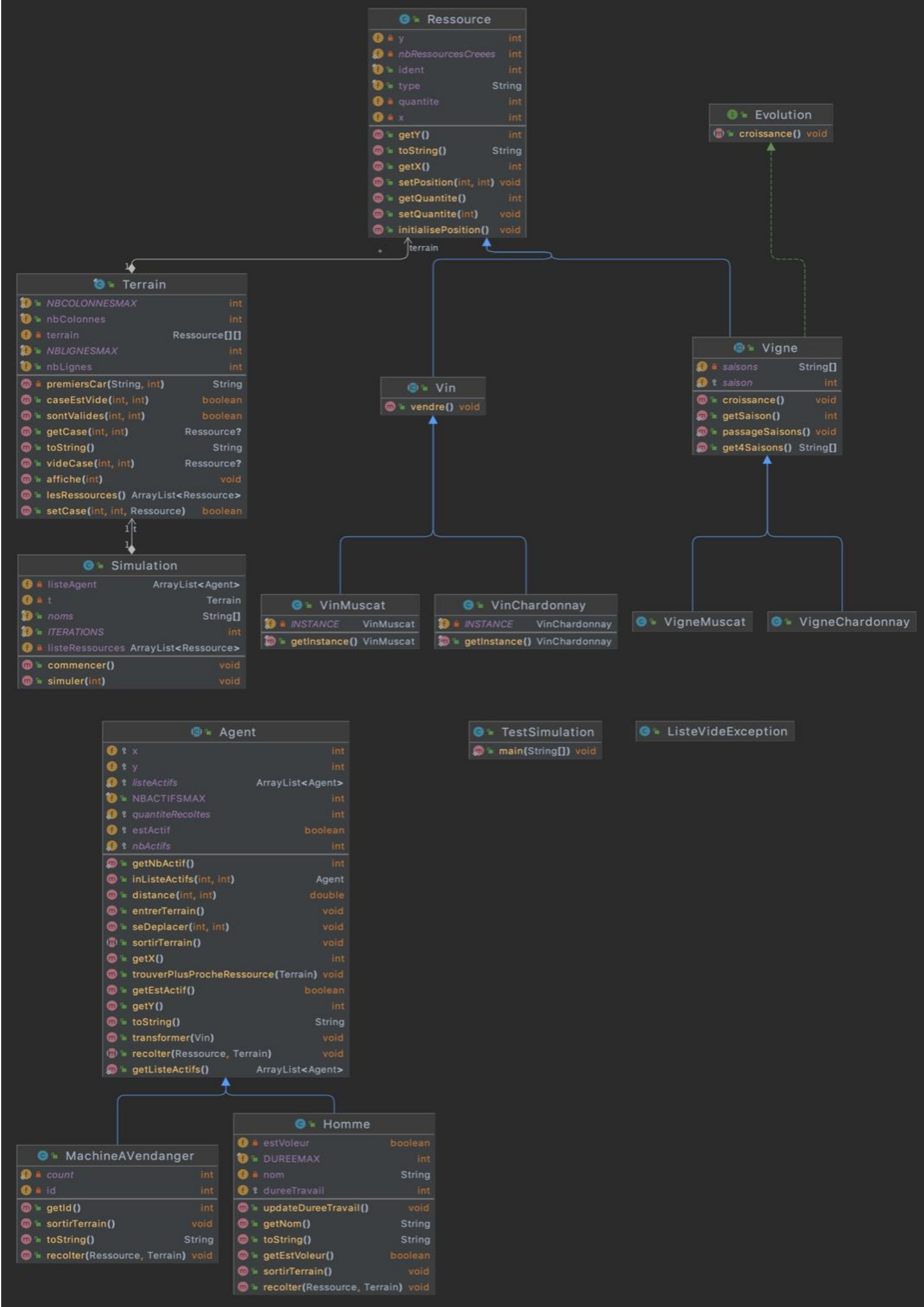
```
      int x = (int)(Math.random()*201);
      this.setQuantite(this.getQuantite() - x);
      System.out.println(x + " bouteilles de " + ((this instanceof VinChardonnay)?"Chardonnay":"Muscat") + " ont été vendues");
    }
    else System.out.println("La récolte de "+((this instanceof VinChardonnay)?"Chardonnay":"Muscat")+" n'a pas été productive
aujourd'hui. Pas de vente :(");
  }
}

----------------------------------------------------------------------------VINCHARDONNAY----------------------------------------------------------------------
public class VinChardonnay extends Vin{
  // Attributs
  private static final VinChardonnay INSTANCE= new VinChardonnay();

  // Constructeurs
  private VinChardonnay(){
    super("Vin Chardonnay");
  }

  // Accesseurs
  public static final VinChardonnay getInstance(){
    return INSTANCE;
  }
}

----------------------------------------------------------------------------VINMUSCAT----------------------------------------------------------------------
public class VinMuscat extends Vin{
  // Attributs
  private static final VinMuscat INSTANCE= new VinMuscat();

  // Constructeurs
  private VinMuscat(){
    super("Vin Muscat");
  }

  // Accesseurs
  public static final VinMuscat getInstance(){
    return INSTANCE;
  }
}

----------------------------------------------------------------------------LISTEVIDEEXCEPTION----------------------------------------------------------------------
public class ListeVideException extends Exception{
  public ListeVideException(){
    super("Liste d'agent ou liste de ressource vide");
  }
}

----------------------------------------------------------------------------SIMULATION----------------------------------------------------------------------
import java.util.*;

public class Simulation{
  // Attributs
  private Terrain t;
  private ArrayList<Agent> listeAgent;
  private ArrayList<Ressource> listeRessources;
  public final static int ITERATIONS = 16; //4 saisons
  public final static String[] noms = {"Elio", "Christina", "Maissa", "Sarah", "Nadia",
                                       "Rayan", "Ina", "Nino", "Dominique", "Siwar", "Lounes",
                                       "Maria", "Paul", "Lea", "Valentin", "Catalina", "Marie"};
  // Constructeur
  public Simulation(int m , int n) throws ListeVideException{
    if (n == 0 || m == 0) throw new ListeVideException();

    // CREATION DU TERRAIN
    t = new Terrain(10,10);

    // CREATION DES RESSOURCES
    for (int i = 0 ; i < m ; i++){
      int ligne = (int)(Math.random()*10);
```

```
    int col = (int)(Math.random()*10);
    if (t.caseEstVide(ligne , col)){
      if ((i % 2) == 0)
        t.setCase(ligne, col, new VigneChardonnay());
      else
        t.setCase(ligne, col, new VigneMuscat());
    }
  }
  listeRessources = t.lesRessources();

  // CREATION DES AGENTS
  listeAgent = new ArrayList<Agent>() ;
  for (int i = 0 ; i < n ; i++){
    if (Math.random()<0.3)
      listeAgent.add(new MachineAVendanger(t));
    else listeAgent.add(new Homme(noms[i%n], t));
  }
}

public void commencer(){
  // AFFICHAGE DE L'ETAT INITIAL DU TERRAIN
  System.out.println("\nBienvenue dans le Vignoble de Christina et Elio ! Bonne Visite :) ");
  t.affiche(16);
  System.out.println("Permettez nous de vous présenter les Vignerons :\n");
  for (Agent a: listeAgent){
    System.out.println(a);
  }
  System.out.println("\nEt maintenant, passons aux choses sérieuses!\n");
}

public void simuler(int i) throws InterruptedException{
    // PROCEDURE
    /* Pour éviter un affichage trop chargé, on se contente de représenter un jour de chaque saison, et de modéliser une partie
    des entrées et sorties sur le terrain. Chaque jour compte 8heures de travail. Chaque incrementation de i est considérée comme
    2heures. On considère que la vente se fait chaque jour. */

  if (i%4==1) System.out.println("\nSaison actuelle: " + (Vigne.get4Saisons())[Vigne.getSaison()] +"\n");

  for (Agent a : listeAgent){ // parcours des agents actifs et inactifs
      a.entrerTerrain(); // on fait entrer un agent en cas de besoin
      if (a.getEstActif()){ //on fait travailler les actifs
        if (t.caseEstVide(a.getX(), a.getY())){
          a.trouverPlusProcheRessource(t);
        }
        a.recolter(t.getCase(a.getX(), a.getY()),t);
        if ( t.getCase(a.getX(), a.getY()) instanceof VigneChardonnay)
          a.transformer(VinChardonnay.getInstance());
        if ( t.getCase(a.getX(), a.getY()) instanceof VigneMuscat)
          a.transformer(VinMuscat.getInstance());
        if (a instanceof Homme && a.getEstActif())
          ((Homme)a).updateDureeTravail();
      }
      a.sortirTerrain();
    }

    if (i%4==0){
      // fin de journee
      (VinChardonnay.getInstance()).vendre();
      (VinMuscat.getInstance()).vendre();
      // Affichage etat actuel
      Thread.sleep(3000);
      t.affiche(16);
      // passage saison
      Vigne.passageSaisons();
      for (Ressource r: listeRessources){
        if (r instanceof Vigne)
          ((Vigne)r).croissance();
      }
    }
}
```

```
}
```

----------TESTSIMULATION----------

```
public class TestSimulation{
  public static void main(String[] args)throws InterruptedException{
    int m = (int)(Math.random()*50);
    int n = (int)(Math.random()*10);
    int i;
    Simulation sim;
    try{
      sim = new Simulation(m , n);
      sim.commencer();
      for (i = 1 ; i <= Simulation.ITERATIONS ; i++)
        sim.simuler(i);
    }
    catch (ListeVideException e){
      System.out.println(e.getMessage());
    }
  }
}
```


Ressource
y int
nbRessourcesCreees int
ident int
type String
quantite int
x int
getY() int
toString() String
getX() int
setPosition(int, int) void
getQuantite() int
setQuantite(int) void
initialisePosition() void
terrain
Evolution
croissance() void
Terrain
NBCOLONNESMAX int
nbColonnes int
terrain Ressource[][]
NBLIGNESMAX int
nbLignes int
premiersCar(String, int) String
caseEstVide(int, int) boolean
sontValides(int, int) boolean
getCase(int, int) Ressource?
toString() String
videCase(int, int) Ressource?
affiche(int) void
lesRessources() ArrayList<Ressource>
setCase(int, int, Ressource) boolean
Vin
vendre() void
Vigne
saisons String[]
saison int
croissance() void
getSaison() int
passageSaisons() void
get4Saisons() String[]
Simulation
listeAgent ArrayList<Agent>
t Terrain
noms String[]
ITERATIONS int
listeRessources ArrayList<Ressource>
commencer() void
simuler(int) void
VinMuscat
INSTANCE VinMuscat
getInstance() VinMuscat
VinChardonnay
INSTANCE VinChardonnay
getInstance() VinChardonnay
VigneMuscat
VigneChardonnay
Agent
x int
y int
listeActifs ArrayList<Agent>
NBACTIFSMAX int
quantiteRecoltes int
estActif boolean
nbActifs int
getNbActif() int
inListeActifs(int, int) Agent
distance(int, int) double
entrerTerrain() void
seDeplacer(int, int) void
sortirTerrain() void
getX() int
trouverPlusProcheRessource(Terrain) void
getEstActif() boolean
getY() int
toString() String
transformer(Vin) void
recolter(Ressource, Terrain) void
getListeActifs() ArrayList<Agent>
TestSimulation
main(String[]) void
ListeVideException
MachineAVendanger
count int
id int
getId() int
sortirTerrain() void
toString() String
recolter(Ressource, Terrain) void
Homme
estVoleur boolean
DUREEMAX int
nom String
dureeTravail int
updateDureeTravail() void
getNom() String
toString() String
getEstVoleur() boolean
sortirTerrain() void
recolter(Ressource, Terrain) void