Return of the zombies!

OMG Zombies!

This is more-or-less what you had to do on paper...

Vous devez compléter un programme en cours de développement simulant une invasion de zombies et vampires, dans un monde peuplé d'humains. Dans ce jeu au tour par tour, toutes les créatures peuvent s'attaquer entre elles ; les vampires peuvent même mordre les humains pour les transformer en l'un des leurs ! Ce n'est qu'une question de temps pour que tous les humains disparaissent... en tout cas, les vivants.

Le code fourni est divisé est cinq classes: le simulateur `Simulator`, et trois classes `Human`, `Zombie` et `Vampire` dérivant toutes d'une classe parente `Character`.

Creez le constructor de la classe `Vampire`. Au début du jeu, les vampires ne sont pas assoifés de sang (c'est-à-dire `Vampire#isThirsty == false`).

Dans la premiere boucle de la methode `Simulator#nextTurn()`, chaque personnage prend par surprise ("encounter") le personnage suivant dans la liste, via la methode `encounterCharacter()`. Modifier le programme afin que, si le personnage A "rencontre" le personnage B :

- si A est un humain, il dit "Go away!"
- si A est un vampire, il attaque B et inflige 10 points de dégats
- si A est un zombie, alors :
  - si B est aussi un zombie, A ne fait rien
  - si B est un humain, A attaque B et inflige 5 points de dégats
  - si B est un vampire, il y a 50% de chances que A attaque B et inflige 5 points de dégats (vous pouvez utiliser la methode statique `Simulator.GenerateRandomBoolean()` pour generer un booleen aleatoire)

  Lorsque A attaque B, il pousse un cri de guerre "B, I'm gonna kill you!" (remplacez B par le nom de l'adversaire), et le nombre de `healthPoints` de B diminue en fonction du nombre de points de dégats infligés.

  Specifications pour cette question:

  - vous **n'êtes pas autorisés** à ajouter une variable d'instance ou de classe, où que ce soit
  - les seules classes que vous êtes autorisés à modifier sont `Human`, `Zombie`, `Vampire`

- vous êtes **autorisés à ajouter des méthodes dans une ou plusieurs classes**. Cependant, **les seules signatures autorisées sont** :

```
public void encounterCharacter(Character c)
protected void attack(Character c)
```

Le non-respect de ces spécifications strictes vaudra zéro pour cette question.

Vous devriez chercher à éviter la duplication de code tout en respectant ces consignes imposées. Et avant de répondre à cette partie, vous devriez lire la suivante...

A présent, un nouveau type de personnage fait son apparition: le zombie sanguinaire. Il est en tout point similaire au zombie classique, mais s'il décide d'attaquer alors il inflige 25 points de dégats (au lieu de 5).

Implémenter la classe `MadZombie` représentant ce personnage, en gardant à l'esprit que vous faites de la programmation orientée objet (donc profitez de l'héritage pour réduire au maximum la taille de cette nouvelle classe!).

Les lignes correspondantes dans la méthode Simulator#init() sont alors décommentées. D'autres modifications sont-elles nécessaires dans le programme pour prendre en compte ces nouvelles créatures ?

Après la première boucle de la méthode `Simulator#nextTurn()`, tous les personnages dont le nombre de points de vie est ≤ 0 doivent être retirés du jeu. Ecrire le code qui retire ces personnages de la liste `Simulator#characterList`.

Après la première série d'attaques, les vampires peuvent aller mordre les humains pour les transformer à leur tour en vampires. Chaque vampire qui est assoifé de sang (c'est-à-dire avec `Vampire#isThirsty == true`) doit appeler la methode `Vampire#bite()` sur le premier humain de la liste qui n'a pas encore été mordu (c'est-à-dire `Human#hasBeenBitten == false`). Ecrire le code correspondant dans la méthode `Simulator#nextTurn()`.

A l'issue de ces morsures, tous les humains ayant été mordus doivent se transformer en vampires. Ces vampires nouvellement créés ont le même nom et le même nombre de points de vie que les humains correspondant au moment de la morsure, et sont assoiffés de sang dès leur création. Remplir le code de la methode correspondante `Human#turnIntoVampire()`, et mettre à jour la liste de personnages dans `Simulator#nextTurn()` : les objets `Human` ayant été mordus doivent être remplacés par les nouveaux objets de type `Vampire` issus de la transformation, à la même position dans `Simulator#characterList`.

Evidemment, aux tours suivants, ces nouveaux vampires pourront à leur tour mordre les humains restants !

Il y a une faille logique dans le design de la methode `nextTurn()`. Parfois des humains censés être morts pourraient tout de même effectuer certaines actions. Saurez-vous la trouver ? Une proposition pour la résoudre ?

OMG the zombies are back, just in time for Christmas!

You're going to make the zombies come back, only more bloodthirsty and vicious, and more realistic than ever (and better-written too!).

Specs

1. Implement more-or-less the original project specifications. Only this time you have more freedom...and more constraints. Use interfaces, abstract classes, enumerations, exceptions where appropriate. This does not mean that you have to implement the classes *exactly* as given above...for example you may have to modify certain methods to make them more easily testable (see below).
    - Note that use of `instanceof` may indicate that the application architecture is not as object-oriented as it could be. Try to minimize use of `instanceof`.
    - You may modify the given classes and their methods as needed.
2. Recycle the graphic interface from Foxes vs. Rabbits for Zombies vs. Vampires vs. Humans vs...
3. There are some behaviour additions:
    - Humans are generally like rabbits, they try to run away from nasty things. But...some locations may contain objects of use to humans which they collect when the go there:
        - a shotgun can stun an attacking zombie for some given number of turns; no effect on vampires though. Can be used until out of ammunition.
        - liquid nitrogen will kill a zombie. Can be used until runs out.
        - a wooden stake will definitively kill a vampire. Can only be used once.
    - Vampires and zombies pursue humans, not each other. Vampires are faster, zombies are slower.
    - All characters are aware of other characters within a limited neighbourhood and act accordingly.
    - Humans breed, unlike vampires and zombies. Humans killed by zombies turn into zombies.
    - Since humans can die of hunger if they're not fed, there's a Helicopter that passes over the field every *N* turns and drops food to selected locations (these may targeted at humans, but there is a random element since dropping food from helicopters can be inaccurate). Helicopters can also drop weapons and ammunition.
4. Testing, testing...Some tests will be supplied. You may want to (or have to!) modify certain methods in order to make them testable – see the DoME postmortem.

Simulator.java

```
package zombiegame;

import java.util.ArrayList;
import java.util.Random;

/**
 * Simulator for Midterm Zombiegame.
 * @author pylaffon
 *
 */
public class Simulator {

    // Default health points for our creatures
    private static final int HP_HUMANS = 100;
    private static final int HP_VAMPIRES = 150;
    private static final int HP_ZOMBIES = 30;

    // List of characters currently in the game
    private ArrayList<Character> characterList;

    /**
     * Initialize game.
     */
    public void init() {
        // Create characters
        Character h1 = new Human("Human 1", HP_HUMANS);
        Character h2 = new Human("Human 2", HP_HUMANS);
        Character v1 = new Vampire("Vampire 1", HP_VAMPIRES);
        Character v2 = new Vampire("Vampire 2", HP_VAMPIRES);
        Character z1 = new Zombie("Zombie 1", HP_ZOMBIES);
        // MadZombie mz1 = new MadZombie("MadZombie 1", HP_ZOMBIES); // uncomment in
                                                                     // question 5b

        // Add characters to the list
        characterList = new ArrayList<Character>();
        characterList.add(h1);
        characterList.add(h2);
        characterList.add(v1);
        characterList.add(v2);
        characterList.add(z1);
        // characterList.add(mz1);    // uncomment in question 5b
    }

    /**
     * Perform all game logic for next turn.
     */
    public void nextTurn() {
        // All characters encounter the next character in the list (question 5)
        for (int i = 0; i < characterList.size(); ++i) {
            Character c = characterList.get(i);
            Character encountered = characterList.get((i+1)%(characterList.size()));
            c.encounterCharacter(encountered);
        }

        // Dead characters are removed from the character list
        // ... add your code here (question 6) ...


        // Each vampire (if he is thirsty) bites the first Human in the list
        // who has not been bitten yet
        // ... add your code here (question 7a) ...
```

```
        // Humans that have been bitten become vampires
        // ... add your code here (question 7b) ...

        // Perform end-of-turn actions for all characters (question 4)
        for (int i = 0; i < characterList.size(); ++i) {
            Character c = characterList.get(i);
            c.endOfTurn();
        }
    }

    /**
     * @return the number of human characters currently in the game
     */
    public int nbHumansAlive() {
        // Need to iterate through the list of characters
        // and count the number of humans
        int nbHumans = 0;
        for (Character character : characterList) {
            if (character instanceof Human) {
                nbHumans++;
            }
        }
        return nbHumans;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {

        // Game initialization
        Simulator sim = new Simulator();
        sim.init();
        System.out.println("Game starts with " + sim.nbHumansAlive() + " humans!");

        // Iterate until no alive human remains
        while (sim.nbHumansAlive() > 0) {
            sim.nextTurn();
        }
        System.out.println("All humans have been eaten!");

    }

    /**
     * Generate a pseudo-random boolean.
     * @return pseudo-random boolean
     */
    public static boolean GenerateRandomBoolean() {
        Random random = new Random();
        return random.nextBoolean();
    }
}
```

Character.java

```
package zombiegame;

/**
 * Parent Character class
 * @author pylaffon
 *
 */
public class Character {

    protected String name;      // name of the character
    protected int healthPoints; // represents the health
                                // (once down to 0, this character will be destroyed)

    /**
     * Constructor of Character class.
     * @param name name of the character
     * @param healthPoints initial HP
     */
    public Character(String name, int healthPoints) {
        this.name = name;
        this.healthPoints = healthPoints;
    }

    // Accessors
    public String getName() {
        return name;
    }
    public int getHealthPoints() {
        return healthPoints;
    }

    /**
     * Decrease the number of HP by a certain amount. HP cannot go below 0.
     * @param reduction number of HP to reduce
     */
    public void reduceHealthPoints(int reduction) {
        healthPoints = healthPoints - reduction;
        if (healthPoints < 0) {
            healthPoints = 0;
        }
    }

    /**
     * Output a character's saying to the screen
     * @param str what the character says
     */
    public void say(String str) {
        System.out.println(name + " says: " + str);
    }

    /**
     * Method triggered when the character described by the current object
     * meets another character, and does something to him (for example, attack).
     * @param c the other character that this character meets
     */
    public void encounterCharacter(Character c) {
        // Default action: do nothing
        System.out.println(name + " meets " + c.name + " and does not attack!");
    }
}
```

Human.java

```
package zombiegame;

/**
 * Human class, derives from Character
 * @author pylaffon
 *
 */
public class Human extends Character {

    private boolean hasBeenBitten;  // false, until a vampire bites this human
    private int turnsSinceLastMeal; // the human will lose health if he's too hungry

    /**
     * Constructor of Human class.
     * At the beginning of the game, humans just had dinner, and have not been bitten yet.
     * @param name name of the character
     * @param healthPoints initial HP
     */
    public Human(String name, int healthPoints) {
        super(name, healthPoints);
        hasBeenBitten = false;
        turnsSinceLastMeal = 0;
    }

    // Accessors and mutators
    public boolean getHasBeenBitten() {
        return hasBeenBitten;
    }

    public void setHasBeenBitten(boolean hasBeenBitten) {
        this.hasBeenBitten = hasBeenBitten;
    }

    /**
     * Method triggered on each character at the end of each turn.
     */
    public void endOfTurn() {
        // Increment the number of turns since the last time the human ate
        turnsSinceLastMeal++;

        // If the human is too hungry, he will lose health...
        if (turnsSinceLastMeal > 3) {
            healthPoints -= 2;
        }
    }

    /**
     * Transform this human who has been bitten, into a blood-thirsty vampire.
     * @return a new object of class Vampire, with the same name and healthpoints
     *         as this human; the new vampire is immediately thirsty
     */
    public Vampire turnIntoVampire() {
        // ... add your code here (question 7b) ...
    }
}
```

Zombie.java

```
package zombiegame;

/**
 * Zombie class, derives from Character.
 * @author pylaffon
 *
 */
public class Zombie extends Character {

    /**
     * Constructor of Zombie class.
     * @param name name of the character
     * @param healthPoints initial HP
     */
    public Zombie(String name, int healthPoints) {
        super(name, healthPoints);
    }

    /**
     * Output a character's saying to the screen
     * @param str what the character says
     */
    public void say(String str) {
        System.out.println(name + " says: BRAIIIIIINS!");
    }

    /**
     * Method triggered on each character at the end of each turn.
     */
    public void endOfTurn() {
        // Do nothing. Zombies are useless anyway...
    }
}
```

Vampire.java

```
package zombiegame;

/**
 * Vampire class, derives from Character.
 * @author pylaffon
 *
 */
public class Vampire extends Character {

    private boolean isThirsty;

    // ... add your constructor code here (question 2) ...

    // Accessors and mutators
    public boolean getIsThirsty() {
        return isThirsty;
    }

    public void setIsThirsty(boolean isThirsty) {
        this.isThirsty = isThirsty;
    }

    /**
     * Method triggered on each character at the end of each turn.
     */
    public void endOfTurn() {
        // The vampire has 50% chance of becoming thirsty, if he is not already
        if (isThirsty || Simulator.GenerateRandomBoolean()) {
            isThirsty = true;
            say("I am thirsty now!!");
        }
    }

    /**
     * Method called when a vampire decides to bite a human
     * @param h Human who gets bitten by this vampire
     */
    public void bite(Human h) {
        // The human has no way to escape. He gets bitten.
        h.setHasBeenBitten(true);
        say("I have bitten you, " + h.getName() + "!");

        // Vampire is not thirsty anymore
        isThirsty = false;
    }
}
```