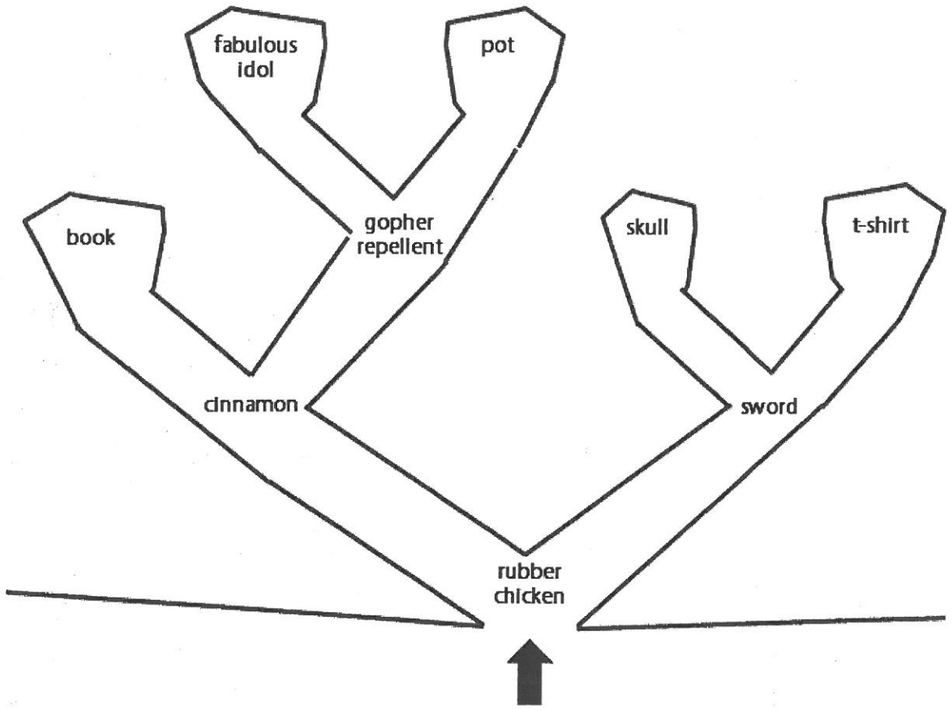

Näiden sääntöjen puitteissa voidaan leikkiin osallistuvalle antaa vaikkapa tällainen ohje: Selvitä, onko luolassa esinettä nimeltä *pot*; takaisinpäin ei saa kääntyä ennen kuin esine on löytynyt tai on osoittautunut, ettei esinettä varmasti ole luolassa.

Tentin liitteessä on annettu koodia, jolla voi kuvata tällaisia luolia. Tehtäväsi on tutustua annettuun koodiin huolellisesti ja täydentää puuttuvat metodit luokista `DeadEnd` ja `Fork` (yhteensä neljä metodia). Näin tulet lisänneeksi ohjelmaan kuvatunlainen esineenetsimistoiminnallisuuden sekä mahdollisuuden selvittää luolan "syvyyden" eli peräkkäisten risteysten määrän.

Ohjeita ja vinkkejä:

- Varmista ensin, että ymmärrät, mitä pitäisi saada aikaan. Tässä auttaa esimerkkiohjelma `DungeonTest`.
- Käytä `Fork`-luokan metodeissa rekursiota (eli kutsu metodia metodista itsestään). **Tämä on tässä paitsi kätevää myös vaatimus!** Inspiraatiota rekursiivisten metodien laatimiseen voi hakea vaikkapa `Fork`-luokan toimivana annetusta `size`-metodista.
- Voit olettaa, että esineet on sijoitettu luolaan aakkosjärjestyksessä (eli tätä ei tarvitse tarkastaa).
- Merkkijonojen "aakkosjärjestyksen" määrittämiseen kelpaavat tässä tutut vertailuoperaattorit kuten < ja >.
- Tentin jälkeen voit halutessasi selvittää Internetistä, mikä on binaaripuu (*binary tree*) ja miten se liittyy tehtävän aiheeseen. Binaaripuista syvällisemmin jatkokursseilla.

Tehtävästä saa palautteena jonkin näistä kolmesta:

- **puutteellinen:** Vastaus on selvästi puutteellinen tai väärin. (Ei arvosanakorotusta.)
- **sinnepäin:** Vastauksessa on kyllä oikeaakin, mutta myös jotain oleellista puuttuu. (Tämäkään ei vielä riitä arvosanakorotukseen.)
- **oikein:** Vastauksessa on kaikki pyydetyt osiot toimiviksi toteutettuina; tässäkään tehtävässä ei kuitenkaan muotoseikoista sakoteta. (Tästä saa +1 kurssiarvosanaan.)

Muista käydä täyttämässä kurssipalautekysely viimeistään sunnuntaina 22.12. kurssisivuilla! Käy vaikka heti tentin jälkeen! (Kurssipalaute on pakollinen kurssin osasuoritus.)

Koodi tehtävään 2

```
object DungeonTest extends App {

  val dungeon =
    new Fork("rubber chicken", new Fork("cinnamon", new DeadEnd("book"),
                                                    new Fork("gopher repellent", new DeadEnd("fabulous idol"),
                                                                                 new DeadEnd("pot"))),
                               new Fork("sword",    new DeadEnd("skull"),
                                                    new DeadEnd("t-shirt")))

  println(dungeon.size)        // prints 9, since there are nine items
  println(dungeon.left.size)   // prints 5, since there are five items to the left of "rubber chicken"
  println(dungeon.right.size)  // prints 3, since there are three items to the right of "rubber chicken"

  println(dungeon.depth)       // should print 3, since the deepest parts ("pot"/"fabulous idol")
                               // can be reached from "rubber chicken" through three forks
  println(dungeon.left.depth)  // should print 2, since starting from "cinnamon" you reach the deepest
                               // parts of that branch ("pot"/"fabulous idol") via two forks
  println(dungeon.right.depth) // should print 1, since starting from "sword", you reach the deepest parts
                               // of that branch ("skull"/"t-shirt") via just that one fork

  println(dungeon.contains("rubber chicken"))       // should print true; there is a rubber chicken right there
  println(dungeon.contains("pot"))                  // should also print true; a pot exists in the dungeon
  println(dungeon.contains("red herring"))          // should print false; no such item can be found
  println(dungeon.right.contains("rubber chicken")) // should print false; only searches the right-hand branch
  println(dungeon.right.contains("t-shirt"))        // should print true

}

/**
 * Represents parts of a treasure hunt dungeon. Each such part contains a single item. There are two kinds
 * of dungeon parts: dead ends and forks. These are represented by the subclasses of this abstract class.
 *
 * @param item the item that is located in this part of the dungeon.
 */
abstract class DungeonPart(val item: String) {

  /** Returns the total number of items located in the branch of the dungeon starting at this part. */
  def size: Int

  /** Returns the depth of branch of the dungeon that starts at this part, that is, the number of forks
   *  you need to take to reach the deepest part of the branch from here. */
  def depth: Int

  /** Determines whether or not the given item exists somewhere in the branch of the dungeon that starts
   *  from this part. */
  def contains(huntedItem: String): Boolean

}

class DeadEnd(item: String) extends DungeonPart(item) {

  def size = 1

  // TODO: missing the methods depth and contains

}

class Fork(item: String, val left: DungeonPart, val right: DungeonPart) extends DungeonPart(item) {

  def size = 1 + this.left.size + this.right.size

  // TODO: missing the methods depth and contains

}
```