Teréz A. Várkonyi **3. assignment/0. Task** 11th February 2019
NEPTUNCODE
nick@inf.elte.hu
Group 0

Task

Create a program to model survival competition of creatures.

The creatures may belong to 3 species (greenfinch, dune beetle, squelchy). Every creature has a name (string) and power (natural number). The creatures (one after the other) pass a racetrack which consists of fields with different types of ground (sand, grass, marsh). When a creature passes on a ground, it may transmute it while its life changes. If the life of the creature falls to zero or less, it dies. Give the name of the creatures who survive the competition.

- ***Greenfinch**: its life increases by one on grass, decreases by two on sand and by one on marsh. It transfmutes marsh to grass.*
- ***Dune beetle**: its life decreases by two on grass, by four on marsh, and increases by three on sand. It transfmutes marsh to grass and grass to sand.*
- ***Squelchy**: its life decreases by two on grass, by five on sand, and increases by six on marsh. It transfmutes grass to marsh.*

Every data is stored in a text file. The first line contains the number of creatures. Each of the following lines contain the data of one creature: one character for the type (G – Greenfinch, D – Dune beetle, S – Squelchy), name of the creature (one word), and the initial level of exhilaration.
In the last line, a natural number is given for the length of the racetrack, followed by IDs of the ground of each part of the track. The IDs: 0 – sand, 1 – grass, 2 – marsh. The file is assumed to be correct.

Analysis[1]

Independent objects in the task are the creatures. They can be divided into 3 different groups: Greenfinches, Dune beetles and Squelchies.

All of them have a name and a power that can be got. It can be examined what happens when they cross a part of the racetrack. Crossing effects the creature and the ground in the following way:

Greenfinch:

ground	power change	ground change
sand	-2	-
grass	+1	-
marsh	-1	grass

[1] This part may be skipped. It is enough to show the tables of transmute in the Planning section.

Dune beetle:

ground	power change	ground change
sand	+3	-
grass	-2	sand
marsh	-4	grass

Squelchy:

ground	power change	ground change
sand	-5	-
grass	-2	marsh
marsh	+6	-

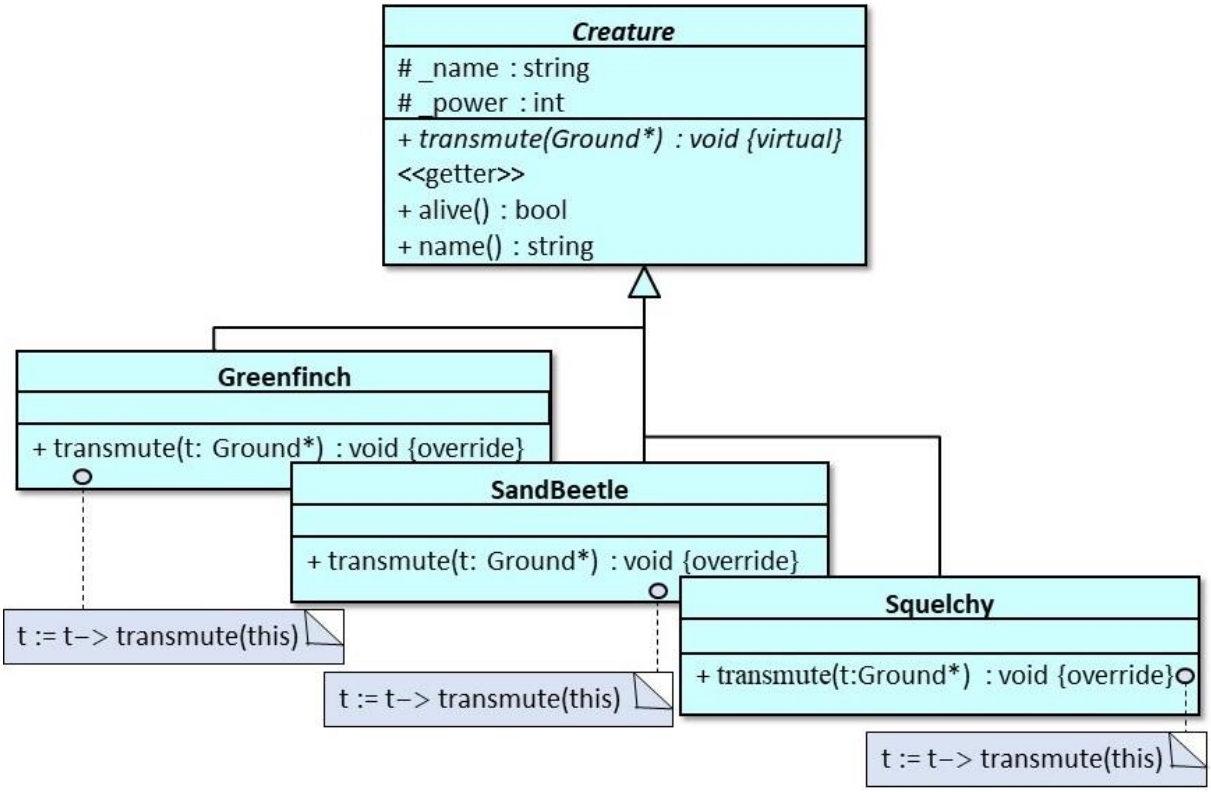
Plan[2]

To describe the creatures, 4 classes are introduced: base class *Creature* to describe the general properties and 3 children for the concrete species: *Greenfinch*, *DuneBeetle*, and *Squelchy*. Regardless the type of the creatures, they have several common properties, like the name (*_name*) and the power (*_power*), the getter of its name (*name()*), if it is alive (*alive()*) and it can be examined what happens when it crosses a ground. This latter operation (*transmute()*) modifies the power of the creature and transmutes the crossed ground. Operations *alive()* and *name()* may be implemented in the base class already, but *transmute()* just on the level of the concrete classes as its effect depends on the species of the creature. Therefore, the general class *Creature* is going to be abstract, as method *transmute()* is abstract and we do not wish to instantiate such class.

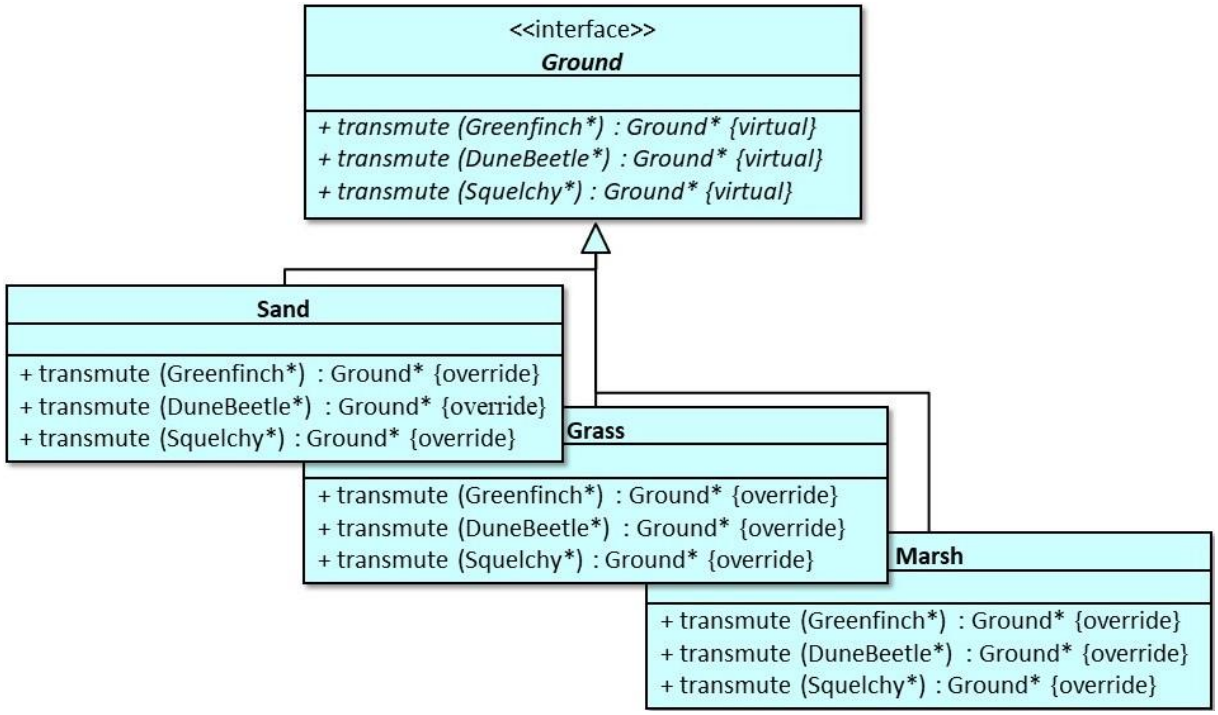
General description of the grounds is done the base class *Ground* from which concrete grounds are inherited: *Sand*, *Grass*, and *Marsh*. Every concrete ground has three methods that show how a Greenfinch, a Squelchy, or a Dune beetle changes during crossing it and how the ground changes, too. Objects are referred by pointers.

The special creature classes initialize the name and the power through the constructor of the base class and override the operation *transmute()* in a unique way. Initialization and the override are explained in Section Analysis. According to the tables, in method *transmute()*, conditionals have to be used in which the type of the ground is examined. Though, the conditionals are not effective if the program might be extended by new ground types, as all of the methods *transmute()* in all of the concrete creature classes have to be modified. To avoid it, design pattern Visitor is applied where the ground classes are going to have the role of the visitor.

[2] Plain text explanation is not necessary for the student documentations



Methods *transmute()* of the concrete creatures expect a ground object as an input parameter as a visitor and calls the methods which corresponds to the species of the creature.



All the classes of the grounds are realized based on the Singleton design pattern, as it is enough to create one object for each class.

In the specification, it is necessary to calculate with the n+1 versions of the track as every creature transmutes it. The 0^{th} version is the initial track. The crossing of one creature is denoted by function *transmute* : $Creature \times Ground^m \rightarrow Creature \times Ground^m$ which gives the changed creature and ground, too. i^{th} version of the track is denoted by $track_i$, which the program is not going to show, it is going to be just a temporal value of variable *track.*

A = $track: Ground^m,\ creatures: Creature^n,\ alive: String^*$

Pre = $creatures = creatures_0 \wedge track = track_0$

Post = $track = track_n \wedge$

$\forall i \in [1..n]: creatures[i], track_i = transmute(creatures_0[i], track_{i-1}) \wedge$

$alive = \bigoplus_{\substack{i=1..n \\ creatures[i].alive()}} < creatures[i].name >$

Concatenation of the creatures (after crossing the track) and transmuting the racetrack step by step are two Summations just as the assortment of the alive creatures. As all of them are based on the same enumerator, they can be merged into the same loop ($i=1$ .. n).

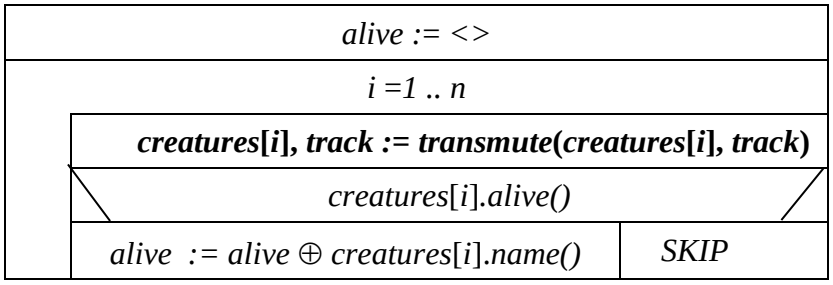
Analogy:

enor(E)	$i = 1$ .. n	
$f(e)$	$transmute(creatures[i], track)_1$	first component of the value of function *transmute()*
s	$creatures$	
H, +, 0	$Creature^*, \oplus, <>$	

enor(E)	$i = 1$ .. n	
$f(e)$	$transmute(creatures[i], track)_2$	second component of the value of function *transmute()*
s	$track$	
H, +, 0	$Creature^*, \ominus, track$	$a \ominus b ::= b$

enor(E)	$i = 1$ .. n
$f(e)$	$<creatures[i]>$ if $creatures[i].alive()$
s	$alive$
H, +, 0	$Creature^*, \oplus, <>$

By merging the above to the same loop, the solution is got:



The i^{th} creature is going to have m+1 states as crosses the track which is, in essence, rebuilt (from $track_{i-1}$ to $track_i$). 0^{th} state of $creatures[i]$ is the given creature ($creatures_0[i]$), while the m^{th} state is the creature after crossing the whole racetrack ($creatures_m[i] = creatures[i]$). The i^{th} creature before crossing the j^{th} ground is denoted by $creature_{j-1}[i]$ from which the j^{th} state is created by method *transmute()* ($creatures_j[i]$) along with the i^{th} state of the track $track_i[j]$.
So, the task to be solved is:

$$\forall j \in [1..m]: creatures_j[i], track_i[j] = transmute(creatures_{j-1}[i], track_{i-1}[j]) \wedge creatures[i] = creatures_m[i]$$

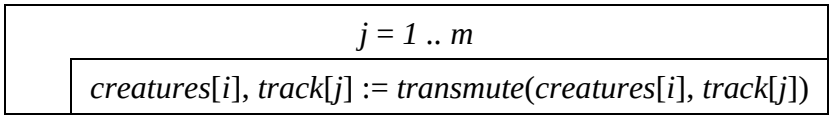
A creature's crossing means changing the state of the creature step by step, while the concatenation of the transmuted grounds is done, too. Both of them are based on Summation with the same enumerator (*j=1 .. m*):

enor(E)	$j = 1 .. m$	
$f(e)$	$transmute(creatures[i], track)_1$	first component of the value of function *transmute()*
s	$creatures[i]$	
H, +, 0	*Creature**, ⊖, $creatures[i]$	$a \ominus b ::= b$

enor(E)	$i = 1 .. n$	
$f(e)$	$transmute(creatures[i], track)_2$	second component of the value of function *transmute()*
s	$track$	
H, +, 0	*Creature**, ⊕, <>	

By merging them into the same loop:

***creatures*[*i*], *track* :=*transmute*(*creatures*[*i*], *track*)**



If it is recognized that neither the creature, nor the grounds change after the death of the creature, the effectiveness of the above algorithm may be improved:

***creatures*[*i*], *track* :=*transmute*(*creatures*[*i*], *track*)**

j := 1	
creatures[*i*].*alive()* ∧ *j* ≤ *m*	
	creatures[*i*], *track*[*j*] := *transmute*(*creatures*[*i*], *track*[*j*])
	j := j+1

Testing

Grey box test cases:

Outer loop (Summation)

1. length-based:
   - zero creature
   - one creature
   - more creatures
2. first and last:
   - first creature survives or not the competition
   - last creature survives or not the competition

Inner loop (Summation)

1. length-based:
   - one creature on a zero-long track
   - one creature on a one-long track transmutes properly
   - one creature on a longer track (survives or dies)
2. first and last:
   - first ground of the track transmutes properly depending on the species of the creature
   - last ground of the track transmutes properly depending on the species of the creature

Examination of function transmute()

Nine different cases depending on the creature and the ground.