```
System.out.println
("Σύγκριση: "  + s1.equals(s2) );       //Εμφανίζει το κείμενο: Σύγκριση: false, αφού τα δύο
                                          string  s1 και s2 δεν έχουν το ίδιο περιεχόμενο.
```

Σημείωση: Δε συγκρίνουμε δύο συμβολοσειρές με τον τελεστή ισότητας ==, ακόμη και αν έχουν το ίδιο περιεχόμενο θα παίρνουμε πάντα σαν αποτέλεσμα false. Επειδή τα δύο string είναι αντικείμενα ο τελεστής == δε θα συγκρίνει το περιεχόμενο τους, αλλά τη διεύθυνση τους στη μνήμη.

3.2 Δημιουργία Αντικειμένων

8η εργαστηριακή άσκηση:

Η δημιουργία αντικειμένων (στιγμιότυπα) των κλάσεων, για να γίνει με αυτοματοποιημένο τρόπο, πρέπει να γράψουμε εντολές της γλώσσας Java, μέσα από τον επεξεργαστή κώδικα. Αυτό στις προηγούμενες ασκήσεις γινόταν κάθε φορά που, είτε ανοίγαμε το σενάριο μας, είτε μεταγλωττίζαμε τον κώδικα μας, με δεξί κλικ πάνω στην κλάση και κατόπιν επιλογή του πλαισίου «new όνομα_κλάσης()».

Για να εμφανίζονται ο βάτραχος και οι μύγες, αυτόματα, μόλις ανοίγουμε το σενάριο μας, πρέπει να προσθέσουμε εντολές στην μέθοδο εκείνη που κατασκευάζει τον Κόσμο FrogWorld. Αυτή η μέθοδος λέγεται **κατασκευαστής (constructor)** και έχει το ίδιο όνομα με την κλάση. Ο κατασκευαστής (constructor) είναι μια ειδικού τύπου μέθοδος, που εκτελείται πάντα και αυτόματα όταν δημιουργείται ένα νέο στιγμιότυπο (αντικείμενο) μιας κλάσης και έχει σκοπό την αρχικοποίηση του αντικειμένου. Κάθε κλάση περιέχει ένα προεπιλεγμένο (default) κατασκευαστή (αυτόματα από τη Java), χωρίς παραμέτρους και χωρίς εντολές στο σώμα του, παρόλο που δεν φαίνεται στον κώδικά της.

Ας ανοίξουμε τον επεξεργαστή του FrogWorld για να εξηγήσουμε τον κώδικα που περιέχει:

```
import greenfoot.*;
public class FrogWorld extends World
{
  public FrogWorld ()
  {
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
  }
}
```

Στην πρώτη γραμμή παρατηρούμε την εισαγωγή όλων των κλάσεων του πακέτου greenfoot, με την εντολή import.

Στην δεύτερη γραμμή δηλώνεται, με τη χρήση της λέξης **extends**, η σχέση της κληρονομικότητας μεταξύ των δύο κλάσεων FrogWorld (υποκλάση) και World (υπερκλάση).

Στην τρίτη γραμμή δηλώνεται ο κατασκευαστής της κλάσης FrogWorld. Όπως βλέπετε ο κατασκευαστής ορίζεται μέσα στην κλάση FrogWorld και έχει το ίδιο όνομα μ' αυτή.

Στην τέταρτη γραμμή, μέσα στη μέθοδο του κατασκευαστή, βλέπουμε την εντολή super(600, 400, 1). Με την εντολή **super** καλούμε τον κατασκευαστή της υπερκλάσης World (από την τεκμηρίωση της κλάσης World βλέπουμε ότι υπάρχει ο κατασκευαστής: World(int worldWidth, int worldHeight, int cellSize)) και ως παραμέτρους δίνουμε το πλάτος (600 κελιά), το ύψος (400 κελιά) και τη διάσταση κάθε κελιού (1 pixel) με σκοπό να δημιουργηθεί το πλέγμα του Κόσμου FrogWorld. Η εντολή super, στην περίπτωση που καλεί τον κατασκευαστή της υπερκλάσης, πρέπει να είναι η πρώτη δήλωση που γράφουμε στον κατασκευαστή της υποκλάσης της.

Τώρα ας δηλώσουμε, μέσα στον κατασκευαστή της FrogWorld, ποια αντικείμενα, των κλά-

σεων που έχουμε ορίσει, θα δημιουργούνται. Για να έχουμε ένα βάτραχο και μια μύγα στο σενάριο μας θα γράψουμε τις εντολές:

```
Frog kermit = new Frog();
Fly myga=new Fly();
```

Ο τελεστής **new** δημιουργεί ένα αντικείμενο της κλάσης, το οποίο και αποτελεί μοναδικό στιγμιότυπο της και επιστρέφει επίσης ένα **δείκτη αναφοράς (reference)** σε αυτό. Ο τελεστής new δεσμεύει επίσης την απαιτούμενη μνήμη για το αντικείμενο που δημιουργείται. Οι μεταβλητές kermit και myga είναι οι **δείκτες αναφοράς** αντίστοιχα στα αντικείμενα που δημιουργήσαμε προηγουμένως. Χρησιμοποιώντας αυτό το δείκτη αναφοράς μπορούμε να προσπελάσουμε το αντικείμενο.

Τα αντικείμενα που έχουν δημιουργηθεί βρίσκονται κάπου στη μνήμη του υπολογιστή. Αν θέλουμε να τα εμφανίσουμε μέσα στο πλέγμα του Κόσμου μας, πάμε στην κλάση World και βλέπουμε αν διαθέτει κάποια έτοιμη μέθοδο που κάνει αυτή τη δουλειά (αν δεν υπάρχει έτοιμη μέθοδος πρέπει να φτιάξουμε εμείς μια, γράφωντας κώδικα). Παρατηρούμε ότι υπάρχει η μέθοδος addObject της κλάσης World, την οποία και θα χρησιμοποιήσουμε. Η μέθοδος αυτή περιέχει τρεις παραμέτρους. Η πρώτη δηλώνει το αντικείμενο και οι δυο επόμενες τις συντεταγμένες που θα τοποθετηθεί. Ο κώδικας της κλάσης FrogWorld γίνεται:

```
public class FrogWorld extends World
{

    /**
     * Constructor for objects of class FrogWorld.
     *
     */
    public FrogWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        Frog kermit= new Frog();
        addObject(kermit, 100, 100 );

        Fly myga = new Fly();
        addObject( myga, 200, 200 );
```

Η παραπάνω εντολή κλήσης της μεθόδου addObject προσθέτει το αντικείμενο kermit στο σημείο του Κόσμου με συντεταγμένες (100,100) και το αντικείμενο myga στο σημείο του Κόσμου με συντεταγμένες (200,200). Η αρχή των αξόνων του Κόσμου μας είναι η πάνω αριστερή γωνία.

Με τον ίδιο τρόπο δημιουργούμε μερικά ακόμα αντικείμενα μύγες (Fly) στο πλέγμα του Κόσμου μας κα τα προσθέτουμε με τη μέθοδο addObject σε διαφορετικές συντεταγμένες του FrogWorld.

Παρακάτω φαίνεται ο κώδικας του κατασκευαστή της κλάσης FrogWorld στον οποίο έχουμε προσθέσει τις απαραίτητες εντολές για την δημιουργία αντικειμένων, ενός βατράχου και τριών μυγών.

```
import greenfoot.*;
public class FrogWorld extends World
{
  public FrogWorld ()
    {
       super(600, 400, 1);
       Frog kermit= new Frog();
       addObject( kermit, 100, 100 );
       Fly myga = new Fly();
```

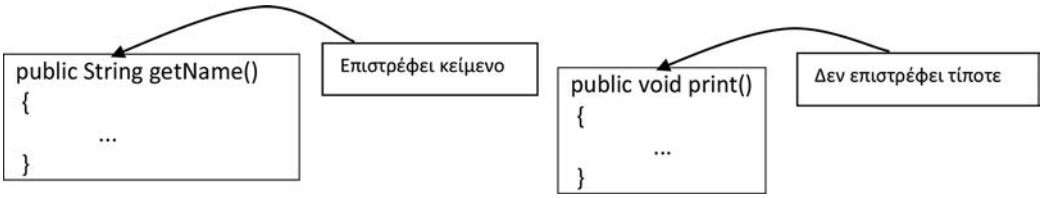
```
        addObject(myga, 200, 200 );
        Fly myga1= new Fly();
        addObject(myga1, 443, 160);
        Fly myga2 = new Fly();
        addObject(myga2, 398, 335);
      }
    }
```

3.3 Κλήση μεθόδων των αντικειμένων

Είδαμε στο προηγούμενο κεφάλαιο ότι οι μέθοδοι στη Java ορίζουν τη συμπεριφορά των αντικειμένων, δηλαδή τις λειτουργίες που μπορεί να εκτελέσει ένα αντικείμενο. Μια μέθοδος είναι ένα μπλοκ κώδικα που έχει ένα όνομα και μπορούμε να την καλέσουμε από οπουδήποτε μέσα από το πρόγραμμα μας (είναι το ισοδύναμο των συναρτήσεων και των υποπρογραμμάτων στον αντικειμενοστρεφή προγραμματισμό).

Στη Java η σύνταξη μιας μεθόδου περιλαμβάνει τα εξής στοιχεία:

- ***Τον προσδιοριστή πρόσβασης***. Μπορεί να είναι **public**, **private** ή **protected** (θα συζητηθούν αργότερα).
- ***Τον τύπο της πληροφορίας που επιστρέφουν (return type)***. Δείχνει τον τύπο δεδομένων της τιμής που επιστρέφεται από την μέθοδο. Π.χ. η μέθοδος int getSpeed() όταν εκτελεσθεί θα επιστρέψει έναν ακέραιο αριθμό που θα μας πληροφορεί για την τρέχουσα ταχύτητα του αντικειμένου. Ένα άλλο παράδειγμα μεθόδου είναι η boolean isAtEdge(), που είναι μια από τις μεθόδους της κλάσης Actor του Greenfoot. Η λέξη boolean μπροστά από το όνομα της μεθόδου σημαίνει ότι η τιμή που επιστρέφει η μέθοδος είναι true ή false και μας πληροφορεί αν το αντικείμενο έχει φτάσει ή όχι στην άκρη του πλέγματος. Αν η μέθοδος δεν επιστρέφει κάποια τιμή τότε χρησιμοποιούμε τη λέξη void (Η λέξη void σημαίνει «τίποτα»). Π.χ.



- ***Το όνομα της μεθόδου.*** Καλό είναι για όνομα να χρησιμοποιούμε μια λέξη που να υποδηλώνει το τι κάνει η συγκεκριμένη μέθοδος. Π.χ. η μέθοδος turn, δίνει εντολή στο αντικείμενο να στρίψει.
- ***Τη λίστα παραμέτρων.*** Αυτές γράφονται μέσα στις παρενθέσεις, μετά το όνομα της μεθόδου. Οι μέθοδοι μπορεί να περιέχουν καμία ή και περισσότερες από μια παραμέτρους. Οι παράμετροι ορίζονται με τον τύπο δεδομένων μπροστά και το όνομα να ακολουθεί και χωρίζονται μεταξύ τους με κόμμα. Αν δεν έχουμε παραμέτρους χρησιμοποιούμε απλά 2 κενές παρενθέσεις.
- ***Το σώμα της μεθόδου*** σε αγκύλες—με τον κώδικα και τις τοπικές μεταβλητές.

Τα τρία στοιχεία της σύνταξης μιας μεθόδου: ο τύπος επιστροφής, το όνομα μεθόδου και η λίστα παραμέτρων ονομάζονται **υπογραφή της μεθόδου** (method signature).

```
[public ή private ή protected] [τύπος επιστροφής] [όνομα μεθόδου](παράμετρος1, ..., παράμετροςN)
{
  σώμα μεθόδου
}
```

Μια ειδική μέθοδος της Java είναι η main η οποία είναι επιφορτισμένη με το να ξεκινάει την εκτέλεση του προγράμματος. Στη Java για να γίνει ένα πρόγραμμα εκτελέσιμο, πρέπει οπωσδήποτε σε κάποιο από τα αρχεία πηγαίου κώδικα, από τα οποία αποτελείται, να υπάρχει τουλάχιστον μία μέθοδος main.

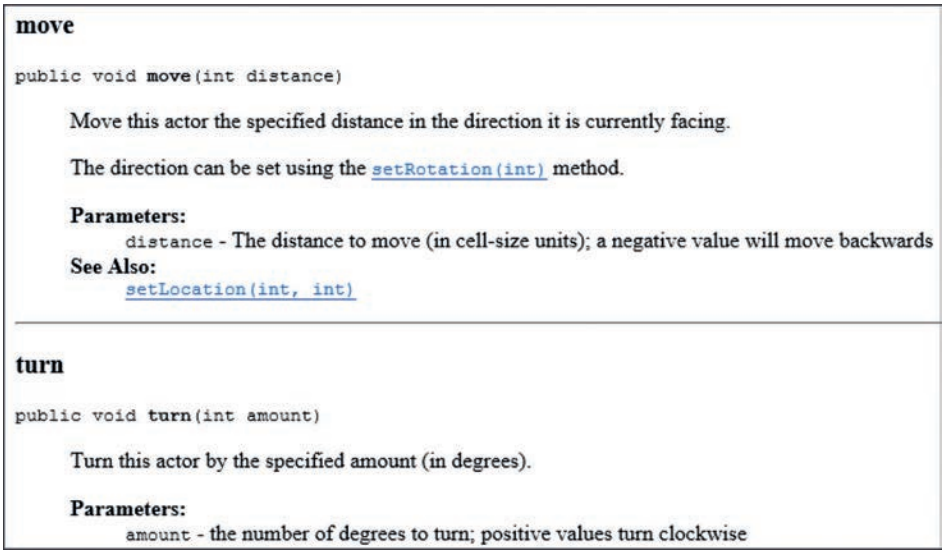
Η σύνταξη της μεθόδου είναι: public static void main(String[] args) { }

Όταν ο διερμηνέας της Java εκτελεί ένα πρόγραμμα ξεκινά από την κλήση της μεθόδου main. Η main κατόπιν καλεί όλες τις υπόλοιπες μεθόδους για να εκτελεστεί το πρόγραμμα.

9η εργαστηριακή άσκηση:

Στο σενάριο FrogEatFlies έχουμε ήδη ένα βάτραχο και μερικές μύγες και μπορούν πλέον να αλληλοεπιδράσουν μεταξύ τους. Η αλληλεπίδραση (interaction) έγκειται κυρίως, στην εκτέλεση εντολών από τα αντικείμενα, μέσα από τη μετάδοση μηνυμάτων. Πρώτα θα γράψουμε κώδικα ώστε όλα τ' αντικείμενα να κινούνται μέσα στο πλέγμα του Κόσμου, χρησιμοποιώντας τις μεθόδους που έχουν κληρονομήσει από την κλάση Actor.

Ανοίγοντας την τεκμηρίωση της κλάσης Actor, βλέπουμε όλες τις μεθόδους της. Οι μέθοδοι που χρειαζόμαστε είναι η *move*, η οποία δίνει σήμα στο αντικείμενο να μετακινηθεί εμπρός σε απόσταση που ορίζουμε στην τιμή που δίνουμε στην παράμετρο της και η *turn*, η οποία λέει στο αντικείμενο να στρίψει κατά γωνία ίση με την τιμή της παραμέτρου της. Μια μικρή περιγραφή για την κάθε μέθοδο βλέπουμε στην Εικόνα 3.3.1, όπως φαίνεται στην τεκμηρίωση της κλάσης Actor.



move

`public void move(int distance)`

Move this actor the specified distance in the direction it is currently facing.

The direction can be set using the `setRotation(int)` method.

Parameters:
`distance` - The distance to move (in cell-size units); a negative value will move backwards

See Also:
`setLocation(int, int)`

---

turn

`public void turn(int amount)`

Turn this actor by the specified amount (in degrees).

Parameters:
`amount` - the number of degrees to turn; positive values turn clockwise

Εικόνα 3.3.1 Οι μέθοδοι move και turn

Στο περιβάλλον του Greenfoot, όταν δημιουργείται μια νέα υποκλάση της κλάσης Actor, προστίθεται αυτόματα στον κώδικα της νέας υποκλάσης η μέθοδος act. Η μέθοδος αυτή είναι αρχικά κενή και μέσα στο σώμα της γράφεται ο κώδικας που καθορίζει τις ενέργειες που πρέπει να κάνει το αντικείμενο.Η μέθοδος αυτή καλείται σε όλα τα αντικείμενα του Κόσμου, που έχουμε δημιουργήσει, κάθε φορά που πατάμε το κουμπί *Δράση* ή *Εκκίνηση* στο περιβάλλον του Greenfoot.

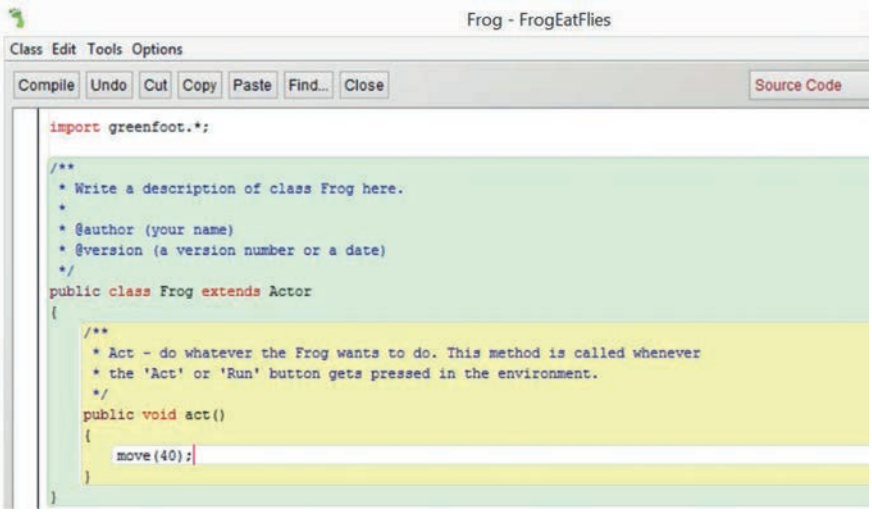
Άρα, τις μεθόδους move και turn θα τις καλούμε μέσα από τη μέθοδο act της κλάσης του αντικειμένου.

Ας δούμε τη μέθοδο move. Η μέθοδος ορίζεται ως εξής:

```
void move( πλήθος βημάτων )
```

Η move έχει ως παράμετρο έναν ακέραιο αριθμό (int) ο οποίος εκφράζει το πλήθος των βημάτων (την απόσταση) που θέλουμε να προχωρήσει το αντικείμενο.

Για παράδειγμα ανοίγουμε τη μέθοδο act του βατράχου (επιλέγοντας *Άνοιγμα επεξεργαστή*) και γράφουμε την εντολή move(40), δηλαδή του δίνουμε την εντολή να μετακινηθεί κατά 40 βήματα. Με όμοιο τρόπο προσθέτουμε μια εντολή move και στην κλάση Fly, αλλά με διαφορετικό πλήθος βημάτων.



Εικόνα 3.3.2 Συγγραφή κώδικα στη μέθοδο act()

Πατώντας το κουμπί ***Εκκίνηση***, κάτω στην οθόνη, εκτελείται διαρκώς (επαναληπτική διαδικασία) η act, για όλα τα αντικείμενα του Κόσμου μας, τα οποία κινούνται με διαφορετική ταχύτητα, ανάλογα με την τιμή της παραμέτρου που έχουν στη move, μέχρι να φτάσουν στην άκρη του Κόσμου. Τ' αντικείμενα στην πραγματικότητα δεν σταματούν με αυτόν τον τρόπο. Εξακολουθούν να προσπαθούν να κινηθούν, αλλά το Greenfoot δεν τ' αφήνει να κινούνται έξω από τον κόσμο (αν τ΄ άφηνε, πώς θα τα μεταφέραμε πάλι πίσω;).

Αλλάζοντας, σε ένα αντικείμενο, την παράμετρο της μεθόδου move με ένα μεγαλύτερο αριθμό η κίνηση θα είναι ταχύτερη (αφού σε κάθε βήμα της επαναληπτικής διαδικασίας θα διανύει μεγαλύτερη απόσταση), ενώ με μικρότερο θα είναι πιο αργή. Μπορείτε να μαντέψετε τι θα συμβεί αν βάλετε έναν αρνητικό αριθμό;

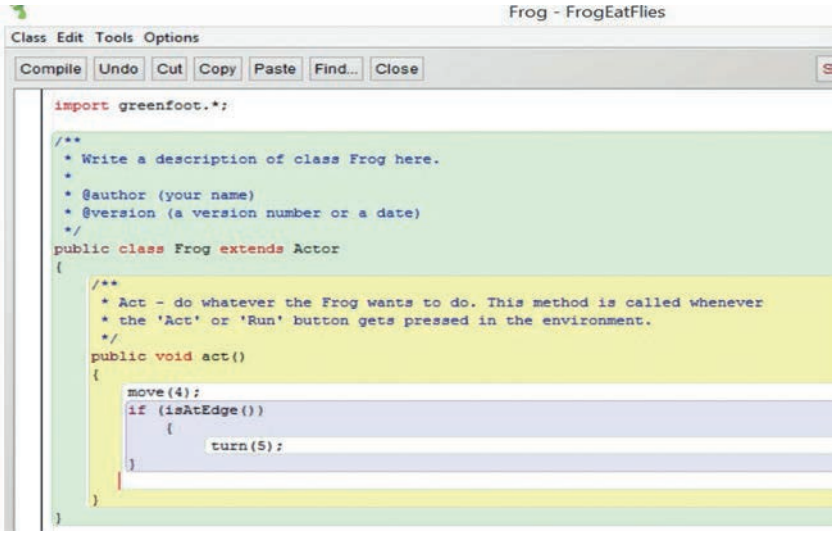
Ας προσθέσουμε μέσα στο σώμα της μεθόδου act και την εντολή *turn*, γράφοντας: turn(5);

Άλλαξε κάτι στην κίνηση του αντικειμένου;

Όσο μικρότερος ο αριθμός στην παράμετρο της μεθόδου turn, τόσο μεγαλύτερο τόξο θα διαγράφει το κινούμενο αντικείμενο (αφού σε κάθε βήμα της επαναληπτικής διαδικασίας θα στρίβει κατά μικρότερη γωνία).

Στη συνέχεια θα κάνουμε τ' αντικείμενα ν' αντιδρούν σε ερεθίσματα από το περιβάλλον τους, παραδείγματος χάρη όταν φτάνουν στην άκρη του Κόσμου να στρίβουν για να συνεχίσουν την κίνηση τους. Η μέθοδος που θα χρησιμοποιήσουμε είναι η isAtEdge(). Η μέθοδος αυτή δεν παίρνει παραμέτρους και επιστρέφει την πληροφορία true ή false, πράγμα που σημαίνει ότι μπορεί να χρησιμοποιηθεί ως συνθήκη μέσα σε μια δομή επιλογής if.

Να γράψετε μέσα στο σώμα της μεθόδου act τον κώδικα, ώστε ο βάτραχος να κινείται ευθεία κατά 4 βήματα τη φορά και **αν** φτάνει στην άκρη του Κόσμου τότε να στρίβει κατά 5 μοίρες. Ο κώδικας στην κλάση Kermit μετά την υλοποίηση των παραπάνω θα γίνει:



Να γράψετε τον αντίστοιχο κώδικα, ώστε η μύγα να κινείται ευθεία κατά 8 βήματα τη φορά και αν φτάνει στην άκρη του Κόσμου τότε να στρίβει κατά 15 μοίρες.

3.4 Κλήση Στατικών Μεθόδων

Στη Java υπάρχουν δύο τύποι μεθόδων: αυτές που ανήκουν στο αντικείμενο, ονομάζονται ***μέθοδοι στιγμιοτύπου (instance methods)*** και καθορίζουν τη συμπεριφορά του αντικειμένου και αυτές που ανήκουν στην κλάση, ονομάζονται ***μέθοδοι κλάσης (class methods)*** και δεν αναφέρονται σε συγκεκριμένο αντικείμενο.

Οι μέθοδοι του αντικειμένου (μέθοδοι στιγμιότυπου – instance methods), μπορούν να κληθούν μόνο μέσω κάποιου αντικειμένου. Η κλήση μιας μεθόδου στιγμιοτύπου συντάσσεται με το όνομα του αντικειμένου ακολουθούμενο από την τελεία και το όνομα της μεθόδου, δηλαδή:

Όνομα αντικειμένου. Όνομα μεθόδου (παράμετροι);

Τέτοια παραδείγματα είναι οι μέθοδοι move και turn, που συναντήσαμε προηγουμένως, οι οποίες δίνουν εντολή σε συγκεκριμένο αντικείμενο να μετακινηθεί ή να στρίψει αντίστοιχα. Αν λοιπόν θέλουμε το αντικείμενο kermit της κλάσης Frog να κινηθεί 100 βήματα θα πρέπει να δώσουμε την εντολή:

kermit.move(100)

Είναι φανερό λοιπόν ότι η χρήση της move χωρίς να αναφερόμαστε σε συγκεκριμένο αντικείμενο δεν έχει νόημα (όταν είμαστε μέσα στην κλάση του αντικειμένου, η αναφορά του ονόματος του είναι περιττή, όπως βλέπουμε και στην Εικόνα 3.3.2).

Πολλές φορές όμως, θέλουμε να σχεδιάσουμε ένα σύνολο μεθόδων οι οποίες δεν είναι συνδεδεμένες με κάποιο αντικείμενο της κλάσης, αλλά υλοποιούν λειτουργίες χρήσιμες για την ίδια την κλάση. Σε αυτή την περίπτωση ορίζουμε μια μέθοδο ως **μέθοδο κλάσης** (class method), χρησιμοποιώντας στην αρχή της δήλωσης της, τη λέξη κλειδί **static**. Οι μέθοδοι κλάσης είναι γνωστές και ως **στατικές μέθοδοι**. Μπορούμε λοιπόν, να καλέσουμε και να χρησιμοποιήσουμε στατικές μεθόδους μιας κλάσης, χωρίς να χρειάζεται να δημιουργηθεί πρώτα κάποιο αντικείμενο αυτής της κλάσης .

Για παράδειγμα, έστω ότι ορίζουμε τη στατική μέθοδο:

```
static int numberOfFrogs( )                    //Δήλωση στατικής μεθόδου η οποία μας
                                                 επιστρέφει το πλήθος των βατράχων που
                                                 υπάρχουν αυτή τη στιγμή στον κόσμο μας
```

Η κλήση της μεθόδου μπορεί να γίνει, είτε από την κλάση, είτε από κάποιο αντικείμενό της κλάσης, όπως φαίνεται παρακάτω:

frogs = Frog.numberOfFrogs() ή frogs = kermit.numberOfFrogs()

όπου το kermit είναι ένα αντικείμενο της κλάσης Frog.

Παράδειγμα: Όλες οι μέθοδοι της κλάσης Math, η οποία βρίσκεται στο πακέτο java.lang, είναι στατικές. Η κλήση τους γίνεται ως εξής: int x = **Math**.max (a, b) ; int y=**Math**.abs(a); int z=**Math**.min(a,b);

Επίσης όλες οι μέθοδοι της κλάσης Greenfoot, η οποία βρίσκεται στο πακέτο greenfoot, είναι στατικές. Η κλήση τους γίνεται ως εξής: x=**Greenfoot**.getRandomNumber(limit);

10η εργαστηριακή άσκηση:

Στο σενάριο μας FrogEatFlies τα αντικείμενα μας κινούνται μέσα στο πλέγμα του Κόσμου, αλλά ή κίνηση τους είναι πάντα η ίδια. Αυτό που θέλουμε στη συνέχεια είναι να προσθέσουμε τυχαία συμπεριφορά στην κίνηση των αντικειμένων μας. Αυτό επιτυγχάνεται στο Greenfoot χρησιμοποιώντας μια μέθοδο που μας παρέχει η κλάση Greenfoot, την getRandomNumber, η οποία επιστρέφει τυχαίους ακέραιους αριθμούς. Η μέθοδος αυτή είναι στατική, δηλαδή δε χρειάζεται να δημιουργηθεί πρώτα κάποιο αντικείμενο αυτής της κλάσης για να τη χρησιμοποιήσουμε.

Η υπογραφή της μεθόδου είναι: int getRandomNumber(int limit)

Η μέθοδος αυτή δέχεται έναν ακέραιο αριθμό (limit) ως παράμετρο, που καθορίζει το όριο των τυχαίων αριθμών που μπορεί να παράγει, δηλαδή παράγει τυχαίους αριθμούς από το 0 έως το limit-1. Π.χ. η getRandomNumber(20) επιστρέφει τυχαίους αριθμούς από το 0 έως 19.

Γράψουμε κώδικα μέσα στη μέθοδο act της κλάσης Frog ώστε, ο βάτραχος να στρίβει κατά 15 μοίρες, μόνο αν ο τυχαίος αριθμός που παράγεται είναι μικρότερος του 10, αλλιώς να συνεχίζει ευθεία και θα στρίβει κατά 5 μοίρες, μόνο όταν φτάνει στην άκρη του Κόσμου. Ο κώδικας έχει την παρακάτω μορφή:

```
public void act()
{
   move(4);
   if (Greenfoot.getRandomNumber(100)<10)
    {
     turn(15);
    }
   if (isAtEdge())
    {
                turn(5);
    }
 }
```

Αφού η getRandomNumber(100) μας δίνει κάθε φορά ένα τυχαίο αριθμό μεταξύ 0 και 99 και δεδομένου ότι οι αριθμοί είναι ομοιόμορφα κατανεμημένοι, στο 10% των περιπτώσεων ο αριθμός που θα παράγεται θα είναι κάτω από 10 και η συνθήκη Greenfoot.getRandomNumber(100)<10 θα είναι αληθής.

Μπορείτε να προγραμματίσετε το βάτραχο ώστε να μη στρίβει κατά 15 μοίρες πάντα, αλλά κατά τυχαία κλίση μέχρι 20 μοίρες; Ποια παράμετρο του προγράμματός σας θα αλλάξετε;

Αντικαταστήστε την εντολή turn(15); με την:

```
int angle = Greenfoot.getRandomNumber(20);
turn(angle);
```

3.5 Δημιουργία των δικών μας μεθόδων

Κατά τη διάρκεια ανάπτυξης μιας εφαρμογής, ορισμένα τμήματα κώδικα (μέθοδοι) χρειάζεται να επαναλαμβάνονται πολλές φορές. Συνήθως, τα τμήματα αυτά αναφέρονται σε λειτουργίες που χρησιμοποιούνται συχνά. Όπως θα δούμε έχουμε τη δυνατότητα να δώσουμε ένα όνομα σε αυτά τα τμήματα κώδικα και να τα καλούμε με αυτό, όποτε τα χρειαζόμαστε. Πρόκειται δηλαδή για μεθόδους που τις δημιουργούμε εμείς για να κάνουν μια συγκεκριμένη δουλειά και αποτελούν μέρη μιας κλάσης.

11η εργαστηριακή άσκηση:

Έως τώρα, στο σενάριο μας, προσθέτουμε διαρκώς γραμμές κώδικα μέσα στη μέθοδο act των κλάσεων μας. Αν συνεχίσουμε έτσι η μέθοδος act θα μεγαλώνει διαρκώς και σε λίγο θα γίνει πολύ δύσκολο να διαβάζουμε τον κώδικα της και να καταλαβαίνουμε τι κάνει. Αν θέλουμε να βελτιώσουμε τον τρόπο που γράφουμε τον κώδικα μας, πρέπει να τον κόψουμε σε μικρότερα μέρη, δημιουργώντας μια μέθοδο για κάθε ομάδα εντολών που επιτελεί μια συγκεκριμένη λειτουργία. Οι νέες μέθοδοι δηλώνονται έξω από το σώμα της μεθόδου act().

Μέχρι τώρα μέσα στη μέθοδο act της κλάσης Frog εκτελούνται 3 διαφορετικές ενέργειες: η κίνηση του βατράχου προς την κατεύθυνση που βρίσκεται, η τυχαία στροφή του και η στροφή του κατά 5 μοίρες όταν φτάσει την άκρη του Κόσμου. Αυτό που θα κάνουμε στη συνέχεια είναι να φτιάξουμε 2 διαφορετικές μεθόδους, μια για κάθε στροφή. Η ενέργεια move(4) δε χρειάζεται να μπει σε ξεχωριστή μέθοδο γιατί είναι μια εντολή μόνο.

Ορίζουμε μια νέα μέθοδο με το όνομα randomTurn, η οποία στρέφει το βάτραχο κάθε φορά κατά τυχαία κλίση μέχρι 20 μοίρες. Έτσι η μέθοδος act της κλάσης Frog μπορεί να ξαναγραφτεί, περιέχοντας λιγότερες λεπτομέρειες, αφού στη θέση του κώδικα της μεθόδου **randomTurn()**, γράφουμε απλώς το όνομα της.

```
public void act()
  {
     move(4);
     randomTurn();

     if (isAtEdge())
      {
                turn(5);
      }
  }
  public void randomTurn()
  {
    if (Greenfoot.getRandomNumber(100)<10)
    {
     int angle = Greenfoot.getRandomNumber(20);
     turn(angle);
    }
  }
```

Δημιουργήστε μια νέα μέθοδο μέσα στην κλάση Frog με όνομα turnAtEdge, χωρίς παραμέτρους. Επιλέξτε το κομμάτι του κώδικα μέσα στη μέθοδο act, όπου γίνεται η στροφή του βάτραχου κατά 5 μοίρες και βάλτε το στη συνέχεια μέσα στη μέθοδο turnAtEdge. Τέλος, κα-

λέστε τη μέθοδο μέσα από την μέθοδο act, γράφοντας απλώς το όνομα της (Μην ξεχάσετε να διαγράψετε το κομμάτι του κώδικα που πήρατε από τη μέθοδο act και το βάλατε μέσα στη μέθοδο turnAtEdge).

Με τη δημιουργία των δύο νέων μεθόδων καταφέραμε δυο πράγματα. Να απλοποιήσουμε αφενός, τον κώδικα της μεθόδου *act*, ώστε να γίνει πιο απλός και κατανοητός για αυτόν που τον διαβάζει και θέλει να καταλάβει τι κάνει και αφετέρου, να υλοποιήσουμε τις μεθόδους randomTurn και turnAtEdge ώστε να είναι ανεξάρτητες από την *act*. Οπότε, αν θέλουμε στη συνέχεια να αλλάξουμε κάτι, όπως π.χ. το αντικείμενο να στρίβει κάθε φορά κατά τυχαία κλίση μέχρι 40 μοίρες και όχι μέχρι 20 μοίρες, αρκεί να αλλάξουμε τον κώδικα μέσα στην randomTurn (αντικαθιστώντας το 20 με το 40), ενώ η act δεν θα χρειαστεί καμία αλλαγή.

Σημείωση: Στη Java τα ονόματα των μεθόδων ξεκινούν με μικρά γράμματα και δεν περιέχουν κενά. Αν το όνομα περιέχει πολλές λέξεις τότε χρησιμοποιούμε κεφαλαίο γράμμα κάθε φορά που αρχίζει μια νέα λέξη π.χ. turnAtEdge.

3.6 Αναφορές και Πέρασμα Παραμέτρων

12η εργαστηριακή άσκηση:

Η μέθοδος turnAtEdge, που δημιουργήσαμε στην προηγούμενη άσκηση, κάνει το βάτραχο να στρίβει κατά 5 μοίρες κάθε φορά που φτάνει στην άκρη του Κόσμου. Αν θέλουμε να στρίβει κατά διαφορετικό αριθμό μοιρών, π.χ. 8 μοίρες, πρέπει να ορίσουμε μια διαφορετική μέθοδο turnAtEdge για κάθε περίπτωση. Αντί να το κάνουμε αυτό μπορούμε να χρησιμοποιήσουμε μία παράμετρο στην μέθοδο, ώστε να μπορεί να χρησιμοποιηθεί κάθε φορά με διαφορετικό αριθμό μοιρών. Περνάμε λοιπόν, τον αριθμό των μοιρών ως παράμετρο της μεθόδου turnAtEdge, ώστε κάθε φορά που την καλούμε να ορίζουμε εμείς το πόσες μοίρες θα στρίβει ο βάτραχος. Για το σκοπό αυτό χρησιμοποιούμε μια ακέραια μεταβλητή την degrees.

```
public void turnAtEdge (int degrees)
{
   if (isAtEdge())
    {
          turn(degrees);
    }
}
public void act()
{
   move(4);
   randomTurn();

   for (int i=0; i<20; i++)
    {
      turnAtEdge( i );
    }
 }
```

Στον παραπάνω κώδικα βλέπουμε ότι, για να εκτελεστεί η μέθοδος turnAtEdge, χρειάζεται να της δώσουμε έναν ακέραιο αριθμό (που είναι η τιμή για την παράμετρο degrees) μέσα στις παρενθέσεις, μετά το όνομα της μεθόδου.

Παρακάτω στον κώδικα μας, χρησιμοποιούμε μιά δομή επανάληψης for όπου η μεταβλητή i παίρνει τιμές από 0 έως 20 και κάθε φορά, στην κλήση της μεθόδου turnAtEdge που γίνεται μέσα στο βρόγχο for, η τρέχουσα τιμή της i περνά ως τιμή της παραμέτρου.

Τις μεταβλητές, που χρησιμοποιούνται στον ορισμό της μεθόδου, τις ονομάζουμε **παραμέτρους** της μεθόδου, ενώ τις τιμές που παίρνουν οι παράμετροι κατά την κλήση της μεθόδου, τις ονομάζουμε **ορίσματα**. Στο παραπάνω παράδειγμα η μεταβλητή degrees είναι η παράμετρος ενώ το i είναι το όρισμα της μεθόδου turnAtEdge.

Η μεταβίβαση παραμέτρων στη Java είναι, όπως λέμε, **κατά τιμή (pass-by-value)** δηλαδή δημιουργούνται αντίγραφα των ορισμάτων, οπότε οποιαδήποτε αλλαγή στις παραμέτρους εντός της μεθόδου δεν έχει καμία επίδραση στα ορίσματα-μεταβλητές που έχουν οριστεί εκτός της μεθόδου. Στο παραπάνω παράδειγμα αυξάνεται κατά ένα η μεταβλητή i όμως η αλλαγή αυτή δεν μεταφέρεται πίσω στην *degrees* η οποία μένει αναλλοίωτη, αφού η i είναι ένα αντίγραφο της degrees.

Στην περίπτωση όμως που η παράμετρος μας δεν αναφέρεται σε βασικό τύπο δεδομένων (int, float, double, κλπ) αλλά σε αντικείμενο κάποιας κλάσης, δεν δημιουργείται αντίγραφο του αντικειμένου. Αυτό συμβαίνει γιατί όταν αναφερόμαστε σε ένα αντικείμενο σύνθετου τύπου η μεταβλητή δεν περιέχει το ίδιο το αντικείμενο αλλά μια διεύθυνση στην περιοχή της μνήμης όπου είναι αποθηκευμένο το αντικείμενο. Η διεύθυνση αυτή λέγεται **αναφορά (reference)** και αποτελεί ουσιαστικά έναν δείκτη στην περιοχή της μνήμης που βρίσκεται το αντικείμενο. Κατά το πέρασμα παραμέτρων δημιουργείται αντίγραφο της αναφοράς και όχι του ίδιου του αντικειμένου, οπότε μέσω της αναφοράς μπορούμε να τροποποιήσουμε το αντικείμενο.

Μέχρι τώρα στο σενάριο μας τα δύο αντικείμενα, ο βάτραχος και η μύγα, δεν αλληλοεπιδρούν μεταξύ τους. Οι βάτραχοι όμως τρώνε τις μύγες, οπότε το επόμενο βήμα που πρέπει να κάνουμε είναι, να βάλουμε τα δύο αντικείμενα ν' αλληλοεπιδρούν, δηλαδή ο βάτραχος μόλις βρει κάποια μύγα να την τρώει.

Οι μέθοδοι της κλάσης Actor, που χρησιμοποιούμε για να προγραμματίσουμε τη συμπεριφορά του βατράχου είναι η isTouching (η οποία ανιχνεύει αν ένα αντικείμενο αγγίζει κάποιο άλλο) και η removeTouching (η οποία εξαφανίζει ένα αντικείμενο όταν το ακουμπήσει κάποιο άλλο). Και οι δύο μέθοδοι δέχονται ως παράμετρο το όνομα μιας κλάσης.

Μέσα στην κλάση Frog ορίζουμε μια νέα μέθοδο με όνομα *eatFlies* που υλοποιεί την ενέργεια που περιγράψαμε παραπάνω.

```
   public void eatFlies()
{
   if (isTouching(Fly.class))
    {
      removeTouching(Fly.class);
    }
}
```

Καλέστε τη μέθοδο eatFlies μέσα από την μέθοδο act.

3.7 Τύποι Επιστροφής

Οι συναρτήσεις σε πολλές γλώσσες προγραμματισμού, όπως στην Pascal και την Python επιστρέφουν ένα αποτέλεσμα με το όνομά τους. Το ίδιο συμβαίνει και με τις μεθόδους της java. Ωστόσο, υπάρχουν μέθοδοι οι οποίοι δεν επιστρέφουν κάποια τιμή, αλλά υλοποιούν κάποια ενέργεια όπως για παράδειγμα την μετακίνηση ενός αντικειμένου. Αυτές οι μέθοδοι που δεν επιστρέφουν τίποτα δηλώνονται με τον τύπο επιστροφής ***void***, όπως για παράδειγμα οι παραπάνω μέθοδοι *act* και *eatFlies*.

13η εργαστηριακή άσκηση:

Στη συνέχεια του σεναρίου μας θέλουμε να προσθέσουμε ένα μετρητή ώστε, κάθε φορά που ο βάτραχος τρώει μια μύγα, να παίρνει ένα πόντο και να κρατάμε το σκορ, το οποίο στη συνέχεια θα εμφανίζεται στην οθόνη μας.

Ανοίγουμε την κλάση Frog για να δηλώσουμε μια μεταβλητή αντικειμένου (Ενότητα 2.1.3) που θα την ονομάσουμε fliesEaten η οποία αποθηκεύει τον αριθμό των μυγών που έφαγε ο βάτραχος. Η τιμή αυτής της μεταβλητής αυξάνεται κατά ένα, όταν εκτελείται η μέθοδος eatFlies.

Ορίζουμε τη νέα μεταβλητή αντικειμένου στην αρχή της κλάσης Frog ως εξής:
private int fliesEaten;

Όπως βλέπουμε ο τύπος της πληροφορίας που αποθηκεύει η μεταβλητή είναι ένας ακέραιος (int) αριθμός.

Η αρχικοποίηση των ακεραίων μεταβλητών γίνεται αυτόματα από τη Java, έτσι η μεταβλητή fliesEaten έχει αρχική τιμή 0. Αν θέλαμε η αρχική της τιμή να είναι διαφορετική από το 0 (π.χ. το 40), τότε θα μπορούσαμε να την αρχικοποιήσουμε χρησιμοποιώντας την έκφραση:

private int fliesEaten = 40;

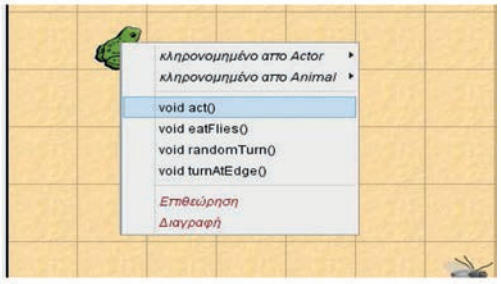
Ο κώδικας που αυξάνει την τιμή της *fliesEaten* γράφεται μέσα στη μέθοδο eatFlies και μέσα στο σώμα της εντολής if. Ο τελεστής ++ μετά το όνομα της μεταβλητής αυξάνει την τιμή της κατά ένα.

Η μέθοδος eatFlies γίνεται:

```
public void eatFlies()
{
   if (isTouching(Fly.class))
    {
      removeTouching(Fly.class);
      fliesEaten ++;
    }
}
```

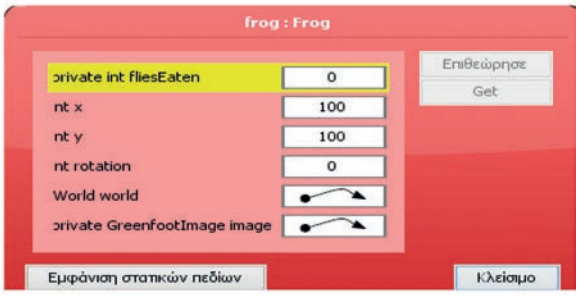
Όταν γράφουμε Fly.class αναφερόμαστε σε ένα αντικείμενο που περιγράφει την κλάση Fly. Αυτό μας δίνει τη δυνατότητα να ελέγξουμε μέσω της isTouching(Fly.class) αν το αντικείμενό μας ακουμπάει ένα αντικείμενο της κλάσης Fly.

Αφού μεταγλωττίσουμε τον κώδικά μας, κάνουμε δεξί κλικ πάνω στο βάτραχο και παρατηρούμε, στο παράθυρο που ανοίγει, ότι υπάρχουν οι μέθοδοι που ορίσαμε μέσα στον κώδικα της κλάσης Frog (Εικόνα 3.7.1).



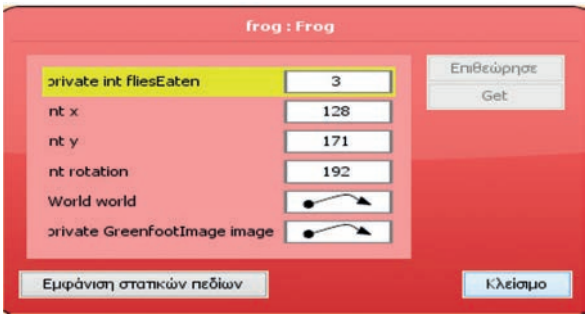
Εικόνα 3.7.1 Μέθοδοι της κλάσης Frog

Επίσης, αν πατήσουμε στο ***Επιθεώρηση*** θα δούμε ότι στις ιδιότητες του αντικειμένου βάτραχος έχει προστεθεί μία ακόμα, η fliesEaten με τιμή 0, αυτή που δηλώσαμε προηγουμένως (Εικόνα 3.7.2).



Εικόνα 3.7.2 Ιδιότητες του αντικειμένου frog

Αν τρέξουμε το πρόγραμμα μας και ο βάτραχος φάει μερικές μύγες, κάνοντας πάλι δεξί κλικ πάνω στο βάτραχο και επιλέγοντας το ***Επιθεώρηση***, θα δούμε ότι η μεταβλητή fliesEaten έχει αποθηκεύσει τη νέα τιμή, δηλαδή τον αριθμό των μυγών που έφαγε ο βάτραχος (Εικόνα 3.7.3).



Εικόνα 3.7.3 Η τιμή της μεταβλητής fliesEaten

Αυτό που χρειαζόμαστε τώρα είναι να εμφανίζουμε στην οθόνη την τιμή της μεταβλητής fliesEaten με τη μορφή ενός μετρητή. Ο μετρητής αυτός θα είναι δυναμικό κείμενο, δηλαδή κείμενο που μεταβάλλεται κάθε φορά που αλλάζει η τιμή της μεταβλητής.

Το κείμενο θεωρείται κι αυτό αντικείμενο και για να μπορέσουμε να το χειριστούμε πρέπει πρώτα να ορίσουμε την κλάση του. Δημιουργούμε μια νέα υποκλάση της *Actor* και την ονομάζουμε Counter, χωρίς να δώσουμε κάποια εικόνα για το αντικείμενο που δημιουργείται από αυτή την κλάση. Αντίθετα, του δίνουμε μια δυναμική εικόνα κατά τη εκτέλεση του σεναρίου, χρησιμοποιώντας τη μέθοδο setImage(GreenfootImage image) της κλάσης Actor. Παρατηρούμε ότι, η παράμετρος που δέχεται αυτή η μεθόδος είναι ένα αντικείμενο εικόνα της κλάσης GreenfootImage.

Η βιβλιοθήκη του greenfoot περιέχει μια κλάση που ονομάζεται GreenfootImage, η οποία παρέχει μεθόδους για το χειρισμό των εικόνων. Οι μέθοδοι της κλάσης GreenfootImage είναι μέθοδοι αντικειμένου, αυτό σημαίνει ότι για να τις χρησιμοποιήσουμε πρέπει να δημιουργήσουμε πρώτα ένα αντικείμενο τύπου *GreenfootImage* χρησιμοποιώντας τη λέξη *new*.

Η κλάση GreenfootImage διαθέτει πέντε διαφορετικούς κατασκευαστές, δηλαδή πέντε διαφορετικές μεθόδους με το ίδιο όνομα για να δημιουργούμε αντικείμενα αυτής της κλάσης (Εικόνα 3.7.4). Οι διαφορετικοί κατασκευαστές διαφοροποιούνται μεταξύ τους ως προς τις παραμέτρους που δέχονται (το πλήθος των παραμέτρων ή/και τον τύπο των παραμέτρων). Αυτό γίνεται όταν θέλουμε ο κατασκευαστής να κάνει διαφορετικές λειτουργίες, ανάλογα με τις παραμέτρους που λαμβάνει, όταν δηλώνουμε τα αντικείμενα μιας συγκεκριμένης κλάσης. Αυτή είναι μια δυνατότητα που μας δίνει η Java και ονομάζεται **υπερφόρτωση (overloading)**.

Constructor Summary

`GreenfootImage(GreenfootImage image)` Create a GreenfootImage from another GreenfootImage.
`GreenfootImage(int width, int height)` Create an empty (transparent) image with the specified size.
`GreenfootImage(java.lang.String filename)` Create an image from an image file.
`GreenfootImage(java.lang.String string, int size, java.awt.Color foreground, java.awt.Color background)` Creates an image with the given string drawn as text using the given font size, with the given foreground color on the given background color.
`GreenfootImage(java.lang.String string, int size, java.awt.Color foreground, java.awt.Color background, java.awt.Color outline)` Creates an image with the given string drawn as text using the given font size, with the given foreground color on the given background color.

Εικόνα 3.7.4 Οι κατασκευαστές της κλάσης GreenfootImage

Μέσα στην κλάση *Counter* δηλώνουμε μια μεταβλητή αντικειμένου που την ονομάζουμε points για ν' αποθηκεύει κάθε φορά την τρέχουσα τιμή του σκορ.

Στον κατασκευαστή (constructor) της κλάσης Counter χρησιμοποιούμε το δεύτερο κατασκευαστή της κλάσης GreenfootImage (Εικόνα 3.7.4) για να δημιουργηθεί μια διαφανή εικόνα, με συγκεκριμένο πλάτος και συγκεκριμένο ύψος και την αποθηκεύουμε στην μεταβλητή με όνομα counterImage. Η εικόνα αυτή θα έχει πλάτος 200 pixels και ύψος 30 pixels. Κατόπιν, με

την κλήση της μεθόδου setImage, αναθέτουμε στον μετρητή (Counter) την εικόνα που δημιουργήσαμε προηγουμένως.

Θέτουμε αρχική τιμή στην μεταβλητή points την τιμή 0 και καλούμε στη συνέχεια μια νέα μέθοδο που ονομάζουμε update().

```
public class Counter extends Actor
{
   private int points;

   public Counter()
   {
      GreenfootImage counterImage=new GreenfootImage(200,30);
      setImage(counterImage);
      points = 0;
      update();
   }
}
```

Η μέθοδος update θα κάνει τις εξής ενέργειες:

1. Παίρνει την τρέχουσα εικόνα του Counter και την αποθηκεύει σε μια μεταβλητή τύπου GreenfootImage που ονομάζουμε img.
2. Καλεί τη μέθοδο clear() για να καθαρίζει την εικόνα του Counter από την προηγούμενη τιμή του.
3. Ορίζει το χρώμα του κειμένου του μετρητή (Counter), χρησιμοποιώντας τη μέθοδο setColor(java.awt.Color color) της κλάσης GreenfootImage. Το χρώμα του κειμένου (η παράμετρος της μεθόδου setColor) προέρχεται από την κλάση Color του πακέτου java.awt, άρα θα πρέπει, όπως είδαμε προηγουμένως, να δηλώσουμε την εισαγωγή της στον κώδικα στην αρχή της κλάσης Counter χρησιμοποιώντας τη λέξη import :

   ```
   import java.awt.Color;
   ```

4. Καλεί τη μέθοδο drawString με παραμέτρους το κείμενο και τις συντεταγμένες της οθόνης που αυτό θα εμφανίζεται.

```
public void update()
{
GreenfootImage img = getImage();
img.clear();
img.setColor(Color.BLACK);
img.drawString("Σκορ:  " + points, 4,20);
}
```

Πριν τρέξουμε την εφαρμογή μας, πρέπει να δημιουργήσουμε στον Κόσμο (FrogWorld), όπως κάναμε και με τα υπόλοιπα αντικείμενα του σεναρίου μας, το νέο αντικείμενο Counter (μετρητής) που θα χρειαστούμε για να κρατάμε το σκόρ, κάνοντας δεξί κλικ πάνω στην κλάση του, επιλέγοντας new Counter() και στη συνέχεια να το σύρουμε στη θέση που επιθυμούμε να εμφανίζεται μέσα στο πλέγμα του Κόσμου. Όπως κάναμε με το βάτραχο και τη μύγα, για να εμφανίζεται ο μετρητής αυτόματα κάθε φορά που εκτελούμε το σενάριο μας, πρέπει να τον δηλώσουμε μέσα στην κλάση FrogWorld. Γράψτε τις εντολές για τη δημιουργία του στον κατασκευαστή της FrogWorld, δίνοντας το όνομα ***metritis*** στη μεταβλητή που θα αποθηκεύσει το αντικείμενο και τις συντεταγμένες x και y του σημείου που θέλετε να εμφανίζεται ο μετρητής μέσα στο πλέγμα του Κόσμου.

Μια ακόμα μέθοδο, που θα χρειαστούμε μέσα στην κλάση Counter, είναι αυτή που θ' αναλάβει ν' αυξάνει τους πόντους του μετρητή και να καλεί στη συνέχεια τη μέθοδο update, ώστε ν' ανανεώνεται η εικόνα του Counter με τη νέα τιμή του σκορ. Αυτή τη μέθοδο θα την ονομάσουμε addScore και θα έχει ως παράμετρο, έναν ακέραιο αριθμό.

```
public void addScore(int value)
{
   points += value;
   update();
}
```

Ο μετρητής είναι έτοιμος. Μεταγλωττίζουμε και τρέχουμε το σενάριο. Το κείμενο του μετρητή εμφανίζεται στην οθόνη, αλλά δεν αυξάνεται η τιμή του όταν ο βάτραχος τρώει κάποια μύγα. Αυτό συμβαίνει γιατί τα δύο αντικείμενα βάτραχος (Frog) και μετρητής (Counter), δεν γνωρίζουν το ένα την ύπαρξη του άλλου και δεν μπορούν να επικοινωνήσουν απευθείας. Το μόνο αντικείμενο στον Κόσμο μας που γνωρίζει (έχει αναφορά) και στα δύο αντικείμενα, είναι ο Κόσμος του σεναρίου μας (FrogWorld).

Αυτό που πρέπει να γίνει είναι να γραφεί ο κώδικας, ώστε όταν ο βάτραχος τρώει μια μύγα να μπορεί να στέλνει ένα μήνυμα στο αντικείμενο Counter για να αυξάνει το σκορ του και να το εμφανίζει. Περνώντας μια αναφορά του αντικειμένου Counter στον κατασκευαστή του αντικειμένου βάτραχος, τα δύο αντικείμενα θα μπορούν να επικοινωνήσουν.

Μέχρι τώρα δεν έχουμε ορίσει κάποιο κατασκευαστή για την κλάση Frog και όπως είδαμε στο κεφάλαιο 3.2, εκτελείται ο default κατασκευαστής του αντικειμένου. Θα δημιουργήσουμε ένα κατασκευαστή για την κλάση Frog που θα εξυπηρετεί τις ανάγκες μας, αφήνοντας προς το παρόν το σώμα του χωρίς κάποια εντολή.

```
public Frog(Counter score)
{
}
```

Πηγαίνουμε στον κατασκευαστή της κλάσης FrogWorld και στο σώμα του αλλάζουμε την εντολή για τη δημιουργία του βατράχου με την παρακάτω, ώστε να δημιουργεί το αντικείμενο βάτραχος με το νέο κατασκευαστή που ορίσαμε προηγουμένως:

Frog kermit= new Frog (metritis);

Η τελική μορφή της κλάσης FrogWorld :

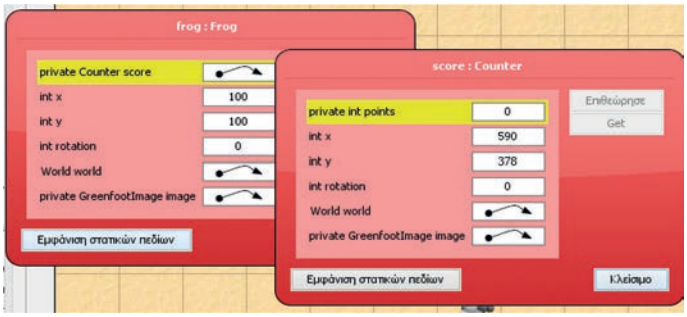
```
public FrogWorld()
  {
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
    super(600, 400, 1);
    Fly myga = new Fly();
    Fly myga1= new Fly();
    Fly myga2= new Fly();
    addObject(myga, 200, 200 );
    addObject(myga1, 443, 160);
    addObject(myga2, 398, 335);

    Counter metritis = new Counter();
    addObject(counter, 590, 390);

    Frog kermit= new Frog(metritis);
    addObject( kermit, 100, 100 );
  }
```

Ο μετρητής counter δημιουργήθηκε και η αναφορά του πέρασε στο αντικείμενο βάτραχος. Τώρα το αντικείμενο βάτραχος «γνωρίζει» το αντικείμενο Counter, με το οποίο σχετίζεται και επιτρέπεται η αποστολή μηνυμάτων από το αντικείμενο βάτραχος προς το αντικείμενο Counter (όχι όμως και το ανάποδο, δεδομένου ότι η συσχέτιση έχει μονόδρομη κατεύθυνση).

Επιστρέφουμε στην κλάση Frog για να γράψουμε τις εντολές (μηνύματα) που θα στέλνει το αντικείμενο βάτραχος προς το αντικείμενο Counter ώστε αυτό ν' αυξάνει την τιμή του.

Πρώτα πρέπει να δημιουργήσουμε μια μεταβλητή, με όνομα score, που θ' αναφέρεται σε ένα αντικείμενο τύπου Counter, όχι όμως απευθείας. Στην μεταβλητή score είναι αποθηκευμένη η διεύθυνση στη μνήμη όπου βρίσκονται τα δεδομένα του αντικειμένου. Αυτή η διεύθυνση ονομάζεται **αναφορά (reference)**. Έτσι θα δώσουμε την εντολή: private Counter score;

Στην Εικόνα 3.7.5 βλέπουμε τις ιδιότητες του βατράχου και φαίνεται καθαρά στην 1η γραμμή η αναφορά στο αντικείμενο τύπου Counter. Πατώντας στο ***Επιθεώρησε*** μας ανοίγει το δεύτερο παράθυρο απ' όπου βλέπουμε τις ιδιότητες του.



Εικόνα 3.7.5 Ιδιότητες του αντικειμένου Frog

Κατόπιν μέσα στο σώμα του κατασκευαστή της κλάσης Frog που δημιουργήσαμε νωρίτερα, αναθέτουμε την παράμετρο (metritis) σ 'αυτό το πεδίο (score). Ο λόγος που το κάνουμε αυτό οφείλεται στο ότι η παράμετρος metritis θα πάψει να υπάρχει, μόλις η εκτέλεση του προγράμματος προχωρήσει στην επόμενη μέθοδο. Γι' αυτό την αποθηκεύουμε σ' ένα πεδίο του αντικειμένου βάτραχος κι έτσι έχουμε μόνιμη αναφορά στο αντικείμενο Counter μέσα από το αντικείμενο βάτραχος.

Η κλάση Frog γίνεται:

```
public class Frog extends Animal
{
   private Counter score;
   public Frog (Counter metritis)
   {
          score= metritis ;
   }
public void eatFlies()
{
   if (isTouching(Fly.class))
    {
      removeTouching(Fly.class);
      score.addScore(1);
       }
}
```

Τώρα η μεταβλητή fliesEaten δεν μας είναι πια απαραίτητη και μπορούμε να τη διαγράψουμε, αφού τον αριθμό των μυγών που τρώει ο βάτραχος τον κρατάει ο μετρητής που δημιουργήσαμε.

3.8. Μέθοδοι και Κληρονομικότητα

14η εργαστηριακή άσκηση:

Θα προσθέσουμε ένα ακόμα αντικείμενο στον Κόσμο, μια πασχαλίτσα (ladybug). Θα ονομάσουμε την κλάση της *Bug* και θα της δώσουμε την εικόνα ladybug1.png. Η κλάση αυτή θα είναι υποκλάση της Animal.

Στη συνέχεια ορίζουμε την μέθοδο act της κλάσης Animal έτσι ώστε το αντικείμενο να κινείται πρώτα πάνω και μετά δεξιά. Η μέθοδος act δεν ορίζεται ξανά στην κλάση Bug, οπότε τα αντικείμενα της Bug διατηρούν τη συμπεριφορά της μεθόδου act της Animal. Σύμφωνα με την ορολογία του αντικειμενοστρεφούς προγραμματισμού η κλάση Bug κληρονομεί από την Animal τη μέθοδο act.

```
import greenfoot.*;
public class Animal extends Actor
{
   public void act()
   {
      move(100);
      turn(90);
      move(100);
   }
}
```

```
import greenfoot.*;

public class Bug extends Animal
{

}
```

Αν τώρα δώσουμε την εντολή act στην πασχαλίτσα αυτή θα κινηθεί νοτιοανατολικά. Στη συνέχεια, ορίζουμε τη μέθοδο act και στην κλάση Bug, χωρίς να πειράξουμε τον κώδικα της κλάσης Animal, έτσι ώστε η πασχαλίτσα να εκτελεί τα ίδια βήματα αλλά με κατεύθυνση βορειοδυτικά.

```
import greenfoot.*;

public class Bug extends Animal
{
   public void act()
   {
      turn(-90);
      move(100);
      turn(-90);
      move(100);
   }
}
```

Τώρα η πασχαλίτσα κινείται βορειοδυτικά, γιατί υπερισχύει η μέθοδος act της κλάσης Bug στην οποία ανήκει, από αυτή της κλάσης Animal.

Το ίδιο συμβαίνει με τις κλάσεις Frog και Fly, όπου υπερισχύει η μέθοδος act όπως την ορίσαμε μέσα στις δικές τους κλάσεις.

Γενικά, όταν δημιουργούμε μια κλάση, ως υποκλάση μιας άλλης, εκτός από τις νέες μεθόδους που προσθέτουμε, επιλέγουμε και ποιες από τις υπάρχουσες μεθόδους θα ορίσουμε ξανά, ώστε να ενεργούν με διαφορετικό τρόπο στην υποκλάση. Με αυτό τον τρόπο εξειδικεύεται η συμπεριφορά των αντικειμένων της νέας κλάσης.

15η εργαστηριακή άσκηση:

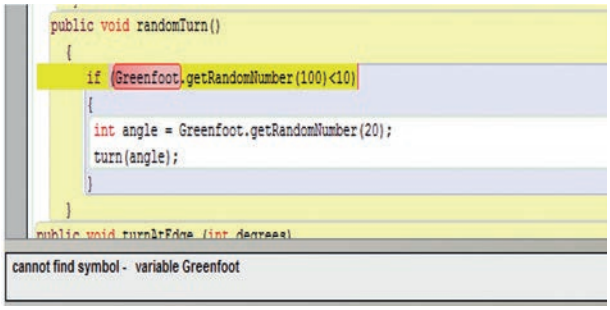
Ας κάνουμε μια μικρή επανάληψη των όσων μάθαμε μέχρι τώρα.

Θα ανοίξουμε την κλάση Frog του σεναρίου μας και θα διαβάσουμε τον κώδικα της. Στην πρώτη γραμμή βλέπουμε την εντολή: import greenfoot.*;

Αυτή η εντολή κάνει τη βιβλιοθήκη των κλάσεων του greenfoot διαθέσιμη για χρήση στην δική μας κλάση (Frog) και πληροφορεί συγχρόνως το μεταγλωττιστή γιά το πού θα βρει τις κλάσεις που θα χρησιμοποιήσουμε στον κώδικα. Ο αστερίσκος στο τέλος δηλώνει ότι θέλουμε να είναι διαθέσιμες όλες τις κλάσεις της βιβλιοθήκης greenfoot κι όχι κάποια συγκεκριμένη κλάση.

Η εντολή import πρέπει να προηγείται όλων των δηλώσεων κλάσεων.

Αν διαγράψουμε αυτή την εντολή και πατήσουμε μεταγλώττιση, τότε θα εμφανιστεί ένα μήνυμα λάθους στη μέθοδο randomTurn, στο σημείο που γίνεται η κλήση της μεθόδου getRandomNumber, η οποία ανήκει σε μια άλλη κλάση, την κλάση Greenfoot της βιβλιοθήκης του greenfoot. Αφού δεν έχουμε δηλώσει στην αρχή ότι θα χρησιμοποιήσουμε μεθόδους που ανήκουν στη βιβλιοθήκη του greenfoot, ο μεταγλωττιστής δεν αναγνωρίζει την εντολή που γράφουμε (Εικόνα 3.8.1).



Εικόνα 3.8.1 Μήνυμα λάθους σε κλήση μεθόδου άλλης κλάσης

Αμέσως μετά ακολουθούν τα σχόλια που δίνουν μια μικρή περιγραφή της κλάσης. Είδαμε ότι τα σχόλια ξεκινούν με /** και τελειώνουν με */. Μπορούμε όμως να χρησιμοποιήσουμε τη διπλή ανάποδη κάθετο (//) για να γράψουμε σχόλια που καταλαμβάνουν μόνο μια γραμμή και μπορούμε να βάλουμε σχόλια μιας γραμμής σε οποιαδήποτε γραμμή του κώδικα.

Ακολουθεί η δήλωση της κλάσης που συντάσσεται συνήθως σύμφωνα με τον εξής κανόνα:

public class «όνομα κλάσης» extends «όνομα υπερκλάσης»

{

Πεδία

Κατασκευαστές

Μέθοδοι

}

Τις λέξεις public class θα τις γράφουμε πάντα σε κάθε κλάση που δημιουργούμε. Η λέξη public ονομάζεται **προσδιοριστής πρόσβασης** και θα μιλήσουμε γι' αυτή αναλυτικότερα στο επόμενο κεφάλαιο. Ακολουθεί το όνομα της κλάσης που το ορίζουμε εμείς. Κατόπιν μπαίνει η λέξη extends και το όνομα της υπερκλάσης, που μας πληροφορεί ότι η κλάση που δημιουργούμε είναι υποκλάση της υπερκλάσης που δηλώνουμε μετά τη λέξη extends.

Τέλος ακολουθεί το ζευγάρι των αγκύλων { και } που ορίζουν το σώμα της κλάσης και εκεί μέσα δηλώνονται οι μέθοδοι.

3.9 Καθοδήγηση από γεγονότα

Όλες οι σύγχρονες εφαρμογές απαιτούν την είσοδο κάποιων δεδομένων από τον χρήστη ή τον έλεγχο (καθοδήγηση) από τον χρήστη κατά την εκτέλεση τους. Αυτό γίνεται κυρίως μέσω του πληκτρολογίου και του ποντικιού. Ειδικά, τα παιχνίδια που ο βαθμός αλληλεπίδρασης με

τον χρήστη είναι πολύ υψηλός, ο χειρισμός των μέσων εισόδου είναι κομβικής σημασίας. Αυτό γίνεται μέσα από τον χειρισμό γεγονότων που προκαλούνται όπως το πάτημα ενός πλήκτρου ή η κίνηση του ποντικιού πάνω από ένα αντικείμενο.

Αν μελετήσουμε την τεκμηρίωση της βιβλιοθήκης του Greenfoot θα βρούμε διάφορες μεθόδους για τον χειρισμό αυτών των γεγονότων όπως είναι οι παρακάτω:

```
public static String getKey()
```
Επιστρέφει το όνομα του πλήκτρου που πατήθηκε τελευταία. Αν δεν έχει πατηθεί κάποιο πλήκτρο, από την τελευταία φορά που κλήθηκε, η getKey επιστρέφει null.

```
public static boolean isKeyDown(String keyName)
```
Ελέγχει αν έχει πατηθεί το πλήκτρο με όνομα keyName και επιστρέφει αντίστοιχα true ή false.

```
public static boolean mouseClicked(Object obj)
```
Ελέγχει αν έχει γίνει κλικ με το ποντίκι στο αντικείμενο obj και επιστρέφει αντίστοιχα true ή false. Αν αντί για obj δώσουμε την τιμή null τότε επιστρέφεται true, αν έχει γίνει κλικ οπουδήποτε.

```
public static MouseInfo getMouseInfo()
```
Επιστρέφει ένα αντικείμενο *MouseInfo* το οποίο περιέχει πληροφορίες για την κατάσταση του ποντικιού. Με την κλήση των μεθόδων *getX(), getY()* ξέρουμε τις συντεταγμένες του σημείου όπου έγινε το κλικ ενώ, η *getActor()* μας επιστρέφει το αντικείμενο πάνω στο οποίο έγινε το κλικ.

16η εργαστηριακή άσκηση:

Παρακάτω, δίνουμε ένα παράδειγμα ελέγχου των κινήσεων του βατράχου μέσω των πλήκτρων και του ποντικιού. Εδώ φαίνεται και η σημασία της μεθόδου act στην εκτέλεση του προγράμματος. Όταν πατήσουμε το ***Εκκίνηση*** η act επαναλαμβάνεται συνέχεια. Έτσι μπορούμε να πατάμε πολλές φορές τα πλήκτρα ↑, ↓, ←, → για να μετακινούμε το βάτραχο. Αν πατήσουμε το ποντίκι σε οποιοδήποτε σημείο μέσα στον Κόσμο, εμφανίζονται οι συντεταγμένες της θέσης που έγινε το κλικ.

Η μέθοδος randomTurn που έκανε το βάτραχο να στρίβει τυχαία μέσα στον Κόσμο δε μας χρειάζεται πια, γιατί την κίνηση του βατράχου θα την ελέγχει πλέον ο χρήστης. Οπότε διαγράφουμε τη μέθοδο randomTurn. Επίσης θα διαγράψουμε και τη μέθοδο turnAtEdge για τον ίδιο λόγο.

Δημιουργούμε μια νέα μέθοδο, με το όνομα checkKeys(), που ελέγχει αν πατήθηκε κάποιο πλήκτρο από το πληκτρολόγιο ή το κουμπί απο το ποντίκι. Η υπογραφή της μεθόδου είναι:

public void checkKeys().

Στο σώμα της μεθόδου ορίζουμε δύο ακέραιες μεταβλητές x και y που αποθηκεύουν την τρέχουσα θέση του βάτραχου στους άξονες x και y. Κατόπιν, προσθέτουμε τη δομή ελέγχου if που ελέγχει αν πατήθηκε κάποιο πλήκτρο και δίνει εντολή στο βάτραχο να κινηθεί προς την ανάλογη κατεύθυνση. Δηλαδή, αν πατηθεί το αριστερό πλήκτρο, η τιμή της θέσης του βατράχου στον άξονα x μειώνεται κατά 4 και ο βάτραχος μετακινείται απο την τρέχουσα θέση του 4 βήματα πίσω, όταν εκτελεστεί η μέθοδος setLocation. Ανάλογες ενέργειες πραγματοποιούνται και με το πάτημα των άλλων πλήκτρων.

Επίσης, προσθέτουμε και τη δομή ελέγχου if που ελέγχει αν πατήθηκε το κουμπί απο το ποντίκι και εμφανίζονται οι συντεταγμένες της θέσης που έγινε το κλικ.

```
import greenfoot.*;

public class Frog extends Animal
{
   public void checkKeys ()
   {
      int x = getX();
      int y = getY();
      if ( Greenfoot.isKeyDown("left") ) {
         x = x - 4;
      }
      else if ( Greenfoot.isKeyDown("right") ) {
         x = x + 4;
      }
      else if ( Greenfoot.isKeyDown("up") ) {
         y = y - 4;
      }
      else if ( Greenfoot.isKeyDown("down") ) {
         y = y + 4;
      }
      setLocation(x,y);

      if (Greenfoot.mouseClicked( null ) ) {
         MouseInfo info = Greenfoot.getMouseInfo();
         System.out.println( info.getX() + "  " + info.getY() );
      }
   }
}
```

Οι getX(), getY(), setLocation() είναι μέθοδοι της κλάσης Actor, που έχει κληρονομήσει η Frog.

Η μέθοδος checkKeys() θα καλείται μέσα από τη μέθοδο act της κλάσης Frog, ενώ θα πρέπει να διαγράψουμε την κλήση των μεθόδων randomTurn() και turnAtEdge () μέσα από την act, αφού η κίνηση του βατράχου θα ορίζεται πλέον από το πληκτρολόγιο.

Το επόμενο βήμα είναι ν' αλλάξουμε τον παραπάνω κώδικα και αντί να εμφανίζονται οι συντεταγμένες της θέσης που έγινε κλικ με το ποντίκι, να εξαφανίζεται το αντικείμενο που γίνεται κλικ πάνω του με το ποντίκι. Αυτό επιτυγχάνεται με τη χρήση μεθόδων που θα βρούμε μέσα στη βιβλιοθήκη του greenfoot.

Οι μέθοδοι που θα χρειαστούμε είναι:

- Απο την κλάση Greenfoot: η mouseClicked που επιστρέφει αληθής (True), αν έχει πατηθεί με το ποντίκι κάποιο αντικείμενο και η getMouseInfo, που επιστρέφει ένα αντικείμενο τύπου MouseInfo με πληροφορίες όπως, το αντικείμενο στο οποίο έγινε κλικ απο το ποντίκι και οι συντεταγμένες στις οποίες έγινε το πάτημα του πλήκτρου του ποντικιού.
- Απο την κλάση MouseInfo: η getActor, που επιστρέφει το αντικείμενο στο οποίο έγινε κλικ με το ποντίκι.
- Απο την κλάση World: η removeObject, που αφαιρεί ένα αντικείμενο από τον Κόσμο. Η υπογραφή της μεθόδου removeObject είναι: void removeObject(Actor object). Η μέθοδος δεν επιστρέφει καμία πληροφορία (void) και δέχεται σαν παράμετρο ένα αντικείμενο τύπου Actor. Το αντικείμενο στην παράμετρο της μεθόδου είναι αυτό που θα αφαιρεθεί από τον Κόσμο.
- Απο την κλάση Actor: η getWorld, που επιστρέφει τον Κόσμο στον οποίο βρίσκονται τα αντικείμενα.

Ο κώδικας που πρέπει να γράψετε είναι:

```
if (Greenfoot.mouseClicked( null ) ) {
    MouseInfo info = Greenfoot.getMouseInfo();
    Actor actor=info.getActor();
   getWorld().removeObject(actor);
}
```

3.10 Πίνακες

Ένας πίνακας είναι ένα αντικείμενο, που έχει την ικανότητα και την ιδιότητα να μπορεί να δέχεται και να αποθηκεύει, σε συνεχόμενες θέσεις στη μνήμη, έναν προκαθορισμένο αριθμό στοιχείων, με την προϋπόθεση ότι όλα ανήκουν στον ίδιο τύπο δεδομένων. Δηλαδή τα στοιχεία που θα αποθηκευτούν θα είναι όλα ακέραιοι αριθμοί, ή όλα δεκαδικοί, ή όλα χαρακτήρες, ή όλα αναφορά σε κάποια αντικείμενα της ίδιας κλάσης κτλ. Το μέγεθος του πίνακα καθορίζεται την ίδια χρονική στιγμή της δημιουργίας του και δεν επιτρέπεται καμία αλλαγή στο μέγεθος του μετά τον αρχικό ορισμό του. Με άλλα λόγια, ένας πίνακας δεν έχει την ικανότητα να αυξομειώνεται δυναμικά ανάλογα με τον αριθμό των στοιχείων του, κατά την διάρκεια εκτέλεσης ενός προγράμματος. Το όνομα ενός πίνακα πάντα συνοδεύεται από τα σύμβολα [και] για να ξεχωρίζει από τον ορισμό οποιασδήποτε άλλης μεταβλητής.

Στο παρακάτω σχήμα φαίνεται ένας πίνακας δέκα θέσεων με ακέραιους αριθμούς. Η αρίθμηση των θέσεων ενός πίνακα ξεκινάει από το 0, όπως συμβαίνει και στις περισσότερες γλώσσες προγραμματισμού.

Για να αναφερθούμε, σε κάθε ένα από τα στοιχεία που είναι αποθηκευμένα στον πίνακα, πρέπει να χρησιμοποιήσουμε τον ανάλογο αριθμό (index) που αντιπροσωπεύει τη θέση τους στον πίνακα.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
0	10	20	30	40	50	60	70	80	90

Ο παραπάνω πίνακας ορίζεται με τις παρακάτω ισοδύναμες εντολές:

```
int  a[ ]  = { 0, 10, 20, 30, 40, 50, 60, 70, 80, 90 };
       ή
int  a[ ]  =  new int[10];
for ( int i = 0;  i < 10;  i++)  {
   a[ i ] = 10 * i;
}
```

Για να αναφερθούμε σε ένα στοιχείο του πίνακα, χρειάζεται να δηλώσουμε το όνομα του πίνακα και τη θέση αυτού του στοιχείου. Για παράδειγμα για να θέσουμε μια νέα τιμή στο στοιχείο στη θέση 2 του πίνακα, δηλαδή στο 3ο στοιχείο του πίνακα την τιμή 23 δίνουμε την εντολή:

```
a[ 2 ] = 23;
```

Αν πάλι θέλουμε να αυξήσουμε την τιμή του στοιχείου a[2] κατά 1 δίνουμε την εντολή

a[2]++; που είναι ισοδύναμη με την εντολή a[2]←a[2] + 1

Δηλαδή μπορούμε να διαχειριζόμαστε κάθε στοιχείο ενός πίνακα όπως και μια μεταβλητή.

Προσοχή!!! Όπως φαίνεται και από το σχήμα, η πρώτη θέση ενός πίνακα είναι η θέση 0. Αυτό σημαίνει ότι όταν ορίζουμε έναν πίνακα 10 στοιχείων, αυτά βρίσκονται στις θέσεις από 0 έως 9.

Άρα το 4ο στοιχείο βρίσκεται στη θέση 3 του πίνακα και το 10ο στη θέση 9.

Γενικά το n-οστό στοιχείο βρίσκεται στη θέση n-1 του πίνακα, δηλαδή είναι το a[n-1] .

Όταν θέλουμε να διαβάσουμε, να τυπώσουμε ή να επεξεργαστούμε έναν πίνακα σαρώνουμε τα στοιχεία του χρησιμοποιώντας ένα for και αυτό επειδή σχεδόν πάντα γνωρίζουμε το πλήθος των στοιχείων του πίνακα.

Στη Java μπορούμε να ορίσουμε πολυδιάστατους πίνακες

```
int  matrix[ ][ ]  =  new int[10][10];
```

Μια σημαντική διαφορά των πινάκων στη java σε σχέση με άλλες γλώσσες προγραμματισμού, είναι ότι στη Java κάθε πίνακας έχει ένα πεδίο με όνομα length το οποίο περιέχει το μέγεθος του πίνακα. Για παράδειγμα η προηγούμενη δομή for μπορεί να γραφτεί και ως εξής:

```
for ( int i = 0;  i < a.length;  i++)  {
    a[ i ] = 10 * i;
}
```

Το παρακάτω τμήμα κώδικα υπολογίζει το άθροισμα όλων των στοιχείων του πίνακα

```
int S = 0;
for ( int i = 0;  i < a.length;  i++) {
    S = S + a[ i ];
}
   String strArray[ ] = new String[10];   // Δεσμεύει δέκα θέσεις
```

Προσοχή!!! Οι δέκα θέσεις του πίνακα που δεσμεύονται αφορούν τη διεύθυνση μνήμης στην οποία βρίσκονται οι συμβολοσειρές. Αρχικά έχουν την τιμή null. Για να τους δώσουμε τιμή θα πρέπει να δημιουργήσουμε ξεχωριστά κάθε αντικείμενο strArray*[i]* χρησιμοποιώντας τον τελεστή *new* όπως στο παρακάτω παράδειγμα:

```
for (int i=0; i<strArray.length; i++)
   strArray[ i ] = new String( "Η θέση μου στον πίνακα είναι " + i );
```

Με αυτόν τον τρόπο αποθηκεύουμε σε κάθε θέση του πίνακα τη συμβολοσειρά που θέλουμε.

17η εργαστηριακή άσκηση:

Επιστρέφουμε στο σενάριο μας FrogEatFlies, όπου θ' αλλάξουμε τον κώδικα της μεθόδου checkKeys της προηγούμενης άσκησης, που περιέχει μια σειρά από if εντολές και θα την υλοποιήσουμε με διαφορετική λογική, χρησιμοποιώντας πίνακες.

Κατ' αρχήν δηλώνουμε στην αρχή της κλάσης Frog ένα πίνακα συμβολοσειρών με το όνομα pliktra, που αποθηκεύει τις διαφορετικές ονομασίες των πλήκτρων:

private String[] pliktra={"up","right","down","left"};

Ας σημειώσουμε εδώ ότι το "up" αντιστοιχεί στον αριθμό (index) της θέσης 0 του πίνακα, ενώ το "left" αντιστοιχεί στον αριθμό (index) της θέσης 3 του πίνακα.

Έτσι, αντί να έχουμε τέσσερεις διαφορετικές δομές ελέγχου if, μπορούμε να καλούμε τον πίνακα pliktra μέσα σ' ένα βρόγχο for που θα περιέχει μόνο μια δομή επανάληψης if, ως εξής:

```
for (int i=0; i<pliktra.length; i++) {
      if (Greenfoot.isKeyDown( pliktra[i]) ) {
```

Τώρα το ερώτημα είναι, πως θα συνδέσουμε τον index για κάθε επιλογή πλήκτρου με την κατάλληλη κίνηση που πρέπει να κάνει ο βάτραχος;

Στην προηγούμενη άσκηση είδαμε ότι η κίνηση του βατράχου καθορίζεται απο την αλλαγή στην τιμή της συντεταγμένης x ή της y κάθε φορά. Στην κίνηση προς τ' αριστερά αφαιρούμε 4 απο το x, ενω στην κίνηση προς τα δεξιά προσθέτουμε 4 στο x. Το αντίστοιχο κάνουμε και με το y. Επίσης, όταν οι τιμές των x και y δεν αλλάζουν, είναι σαν να προσθέτουμε στην τιμή τους το μηδέν. Συνοψίζοντας τα παραπάνω, έχουμε διαφορετικές τιμές που πρέπει να προσθέσουμε στο x και στο y και αυτές διαφέρουν ανάλογα με την κατεύθυνση που πρέπει να κινηθεί ο βατράχος. Άρα, θα χρειαστούμε δύο πίνακες. Ένα πίνακα που αποθηκεύει τις τιμές που προσθέτουμε στο x, για κάθε κατεύθυνση και ένα πίνακα που αποθηκεύει τις τιμές που προσθέτουμε στο y, για κάθε κατεύθυνση.

Οι δύο πίνακες θα δηλωθούν κι αυτοί στην αρχή της κλάσης Frog, ως εξής:

```
private int[] xOffset={0,4,0,-4};
private int[] yOffset={-4,0,4,0};
```

Τέλος, μέσα στο σώμα της if θα καλούμε τη μέθοδο setLocation, που θα προσθέτει την κατάλληλη τιμή απο τους δύο πίνακες, στην τιμή των x και y αντίστοιχα.

Ο κώδικας της κλάσης Frog μετά την αλλαγή της μεθόδου checkKeys θα γίνει:

```
public class Frog extends Actor
{
    private  String[] pliktra={"up","right","down","left"};
    private int[] xOffset={0,4,0,-4};
    private int[] yOffset={-4,0,4,0};

    /**
     * Act - do whatever the Frog wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        checkKeys();

    }

    public void checkKeys ()
    {
        for (int i=0; i<pliktra.length; i++) {
            if (Greenfoot.isKeyDown(pliktra[i])) {

                setLocation(getX()+xOffset[i],getY()+yOffset[i]);
            }
        }

        /
        if (Greenfoot.mouseClicked( null ) ) {
            MouseInfo info = Greenfoot.getMouseInfo();
            Actor actor=info.getActor();
            getWorld().removeObject(actor);
        }
    }

}
```