Programare orientată pe obiecte

#4 JAVA
Greenfoot (partea a III-a)

Adrian Runceanu
www.runceanu.ro/adrian

Curs 4

Mediul de dezvoltare vizuala GREENFOOT (continuare)

1. Utilizarea generarii numerelor aleatoare (aranjarii intamplatoare) si intelegerea notatiei Dot Notation si a Constructorilor

2. Definirea metodelor

Metoda getRandomNumber

- Metoda getRandomNumber este o metoda care returneaza un numar aleator situat intre 0 si un parametru limita
- Aceasta metoda este utilizata pentru a elimina anticiparea in programul dumneavoastra
- Prototipul metodei:

```
public static int getRandomNumber(int limit)
```

Tehnica Dot Notation

- Noile subclase pe care le creeati, nu mostenesc metoda getRandomNumber
- Aceasta metoda trebuie sa fie apelata utilizand notatia Dot Notation

Exemplu:

```
Greenfoot.getRandomNumber(20);
```

Cand doriti sa folositi o metoda care nu este mostenita de subclasa pe care o programati, specificati clasa sau obiectul care are metoda, dupa aceea un punct si dupa numele metodei. Aceasta tehnica se numeste Dot Notation.

Tehnica Dot Notation

Formatul Dot Notation include:

- Numele clasei sau obiectului caruia apartine
- Punct
- Numele metodei de apelat
- Lista de parametrii
- Punct si virgula

```
class-name.method-name (parameters);
object-name.method-name (parameters);
```

Tehnica Dot Notation

Exemple de Dot Notation

Metoda getRandomNumber este exemplificata mai jos:

- Apeleaza un numar oarecare de la 0 pana la 15 – exclusiv 15
- Metoda returneaza un numar oarecare intre 0 si 14

```
Greenfoot.getRandomNumber(15)
```

Greenfoot API

- Face referire la Interfata Programelor de Aplicatie GREENFOOT(API) pentru a examina metodele suplimentare ce pot fi apelate folosind Dot Notation.

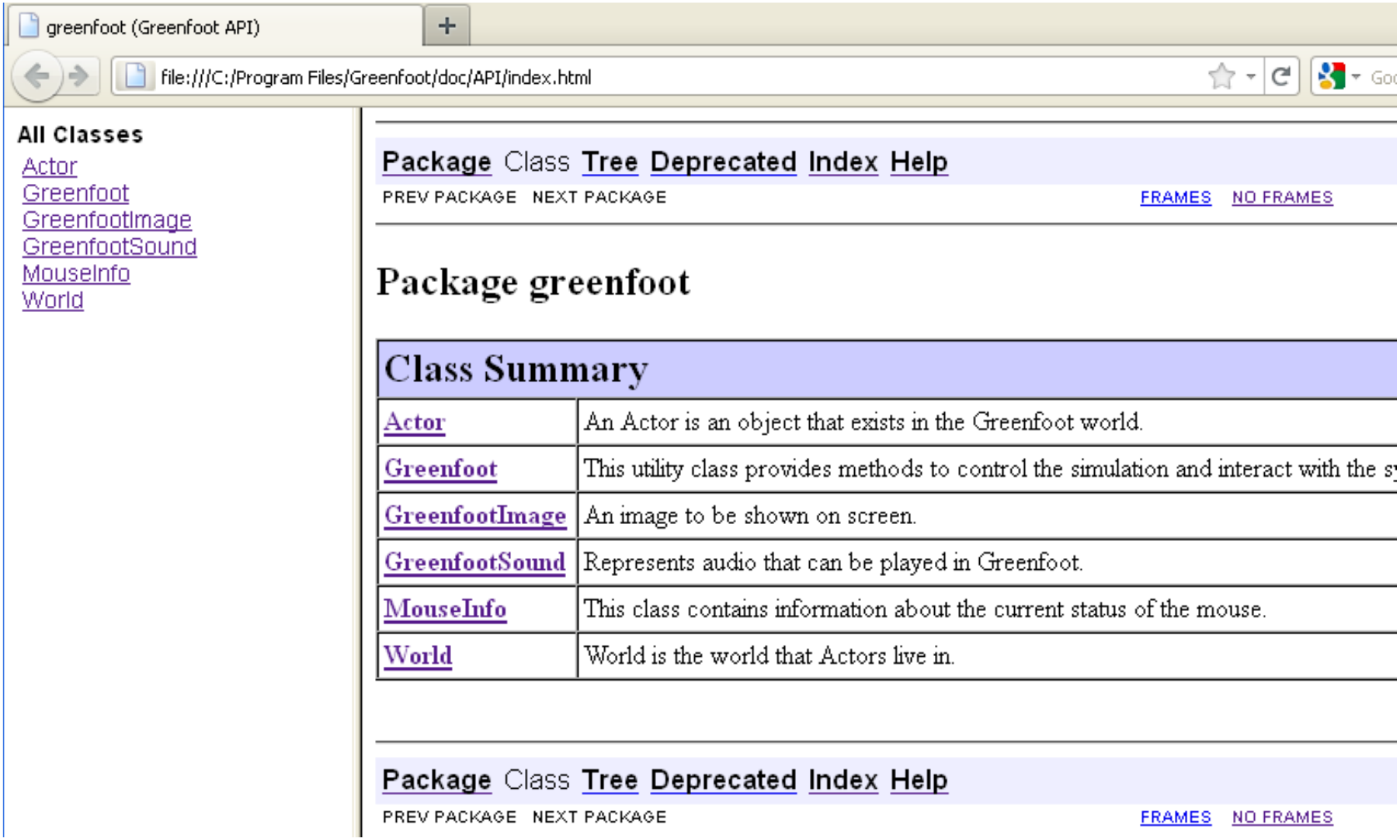
- Interfata Programelor de Aplicatie GREENFOOT(API) afiseaza toate clasele si metodele disponibile in Greenfoot

Greenfoot API

Pasii pentru a vedea metodele din mediul Greenfoot:

1. In mediul Greenfoot, selectati Meniul Help (Ajutor)
2. Selectati Greenfoot Class Documentation (Documentatia Clasei Greenfoot)
3. Dati click pe clasa Greenfoot
4. Analizati semnaturile (particularitatile) si descrierile metodei

Interfata Programelor de Aplicatie GREENFOOT(API)



Operatorii de Comparare

- Folositi operatorii de comparare pentru a compara a valoarea aleatorie cu o valoare stabilita intr-o comanda controlata.
- Exemplul de mai jos determina daca un numar oarecare este mai mic decat 20. Daca da, atunci obiectul se intoarce cu 10 grade.

```
if (Greenfoot.getRandomNumber(100) < 20)
{
    turn(10);
}
```

Operatorii de Comparare

- Operatorii de comparare sunt simboluri care compara doua valori

Symbol	Description
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
==	Equal
!=	Not equal

Probleme de joc rezolvate cu comportament (caracter) aleator

Problema:

Un obiect banana ar trebui sa se miste oricum pentru a fi mai dificil pentru obiectul controlat de la tastatura – maimuta – sa il manance.

Solutia:

- ✓ Banana ar trebui sa se intoarca putin cand se misca.
- ✓ Solutia din cod este sa intoarceti banana cu un numar oarecare de grade (pana la 20) de 6% ori din cate ori se misca.

```
if (Greenfoot.getRandomNumber(100) < 6)
{
    turn(Greenfoot.getRandomNumber(20));
}
```

Formatul Comportamentului Aleator

Codul programului include:

- if (conditia pentru apelul metodei getRandomNumber):
  - parametrul limita 100
  - operatorul de comparare ‘<’
  - Numarul 6 pentru a limita valorile intre 0 si 5
- Corpul propriu-zis al metodei va verifica afirmatia pentru a indica daca obiectul trebuie sa se intoarca pana la 20 de grade doar daca conditia este adevarata.

```
if (Greenfoot.getRandomNumber(100) < 6)
{
    turn(Greenfoot.getRandomNumber(20));
}
```

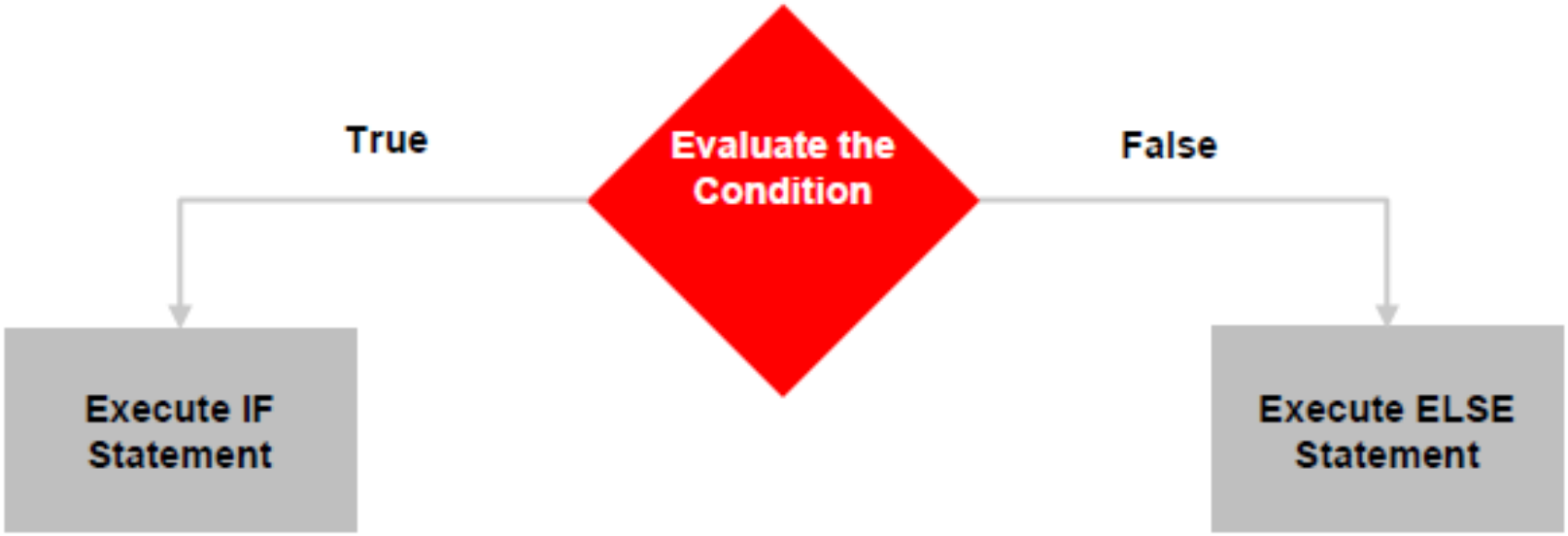
Comportamentul Conditional

Exemplul poate fi programat pentru a prezenta un comportament specific daca o conditie nu este indeplinita, folosind structura IF-ELSE.

Daca exemplul este programat sa se intoarca 6% din timp, ce face acesta in restul de 94% ?

O instructiune IF-ELSE executa prima secventa de cod daca conditia este adevarata si a doua secventa de cod doar daca conditia este falsa, dar nu pe amandoua odata

Executia unei conditii IF-ELSE



Formatul instructiunii IF-ELSE

```
if (condition)
{
     statements;
}
else
{
     statements;
}
```

Exemplu de intructiune IF-ELSE

Daca un numar oarecare intre 0-6 este selectat, se intoarce cu 10 grade. Daca nu, cu 5 grade.

```
if (Greenfoot.getRandomNumber(100) < 7)
{
    turn(10);
}
else
{
    turn(5);
}
```

Crearea automata de instante ale claselor

- Folosind subclasa WORLD, instantele clasei Actor pot fi programate sa apara automat in world(‘lume’) atunci cand un scenariu este initializat

- In Greenfoot, comportamentul standard al instantelor este determinat astfel:
  - ✓ Subclasa World isi adauga automat instantele in world(‘lume’) dupa compilarea si initializarea scenariului
  - ✓ Subclasa Actor trebuie sa aiba instatele adaugate manual de catre jucator

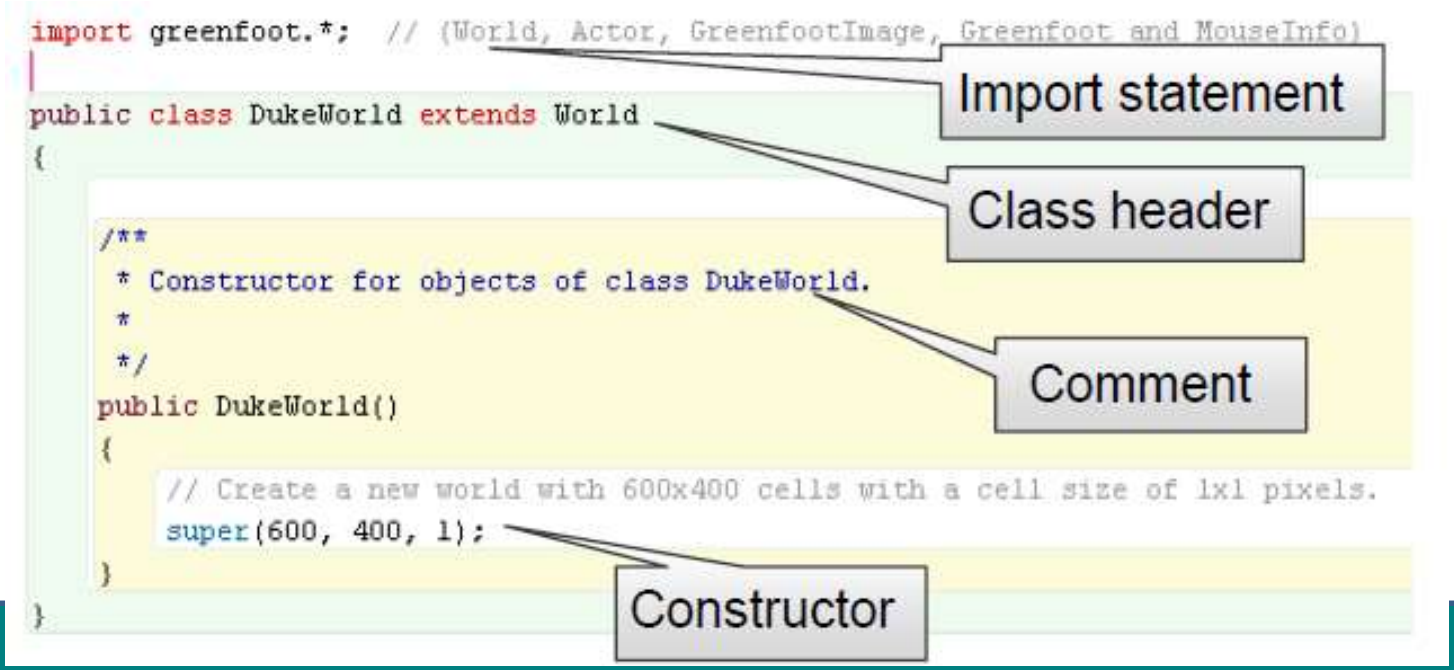
Crearea automata de instante ale claselor

Problema:
Cand un scenariu Greenfoot (cum ar fi frunzele sau persoanele din cadru) este inceput, instanta unei clase trebuie adaugata manual de catre jucator pentru a putea juca jocul.

Solutie:
Programati instante ale claselor care sa fie adaugate automat in lumea jocului atunci cand scenariul este initializat

Codul sursa al Clasei World

- Pentru a intelege crearea automata a instantelor clasei Actor, trebuie sa intelegi cum este structurat codul clasei World.
- Constructorul World este folosit pentru automatizarea instantei Actor cand un scenariu este initializat



Constructori

- ✓ Definesc marimea si rezolutia instantei
- ✓ Nu au nicio valoare de returnare
- ✓ Au acelasi nume ca numele clasei

De exemplu, un constructor World se numeste World.

Un constructor este o metoda speciala care este executata automat atunci cand o instanta a clasei este creata.

Exemplu de constructor al clasei World

Constructorul din exemplul de mai jos este din superclasa World:

- ✓ Marime: x=600, y=400
- ✓ Rezolutie: 1 pixel/celula

Cuvantul cheie ‘super’ in corpul constructorului apeleaza superclasa World pentru fiecare instanta a subclasei DukeWorld.

```
public DukeWorld()
{
      super(600, 400, 1);
}
```

Size

Resolution

Crearea automata a instatelor clasei Actor

- ✓ Acest constructor al clasei World adauga un obiect Duke de coordonate(X,Y), specificate folosind metoda addObject

```
public DukeWorld()
{
      super(560, 560, 1);
      addObject (new Duke(), 150, 100);
}
```

Metoda addObject

- ✓ Metoda addObject este o metoda a clasei World care adauga un nou obiect pe coordonatele x, y date
- ✓ Cuvantul cheie care trebuie mentionat pentru a crea un obiect nou intr-o anume clasa este ‘new’

Parametrii metodei:
- numele obiectului dintr-o clasa Actor
- o variabila de tip intreg ca si coordonata X
- o variabila de tip intreg ca si coordonata Y

Prototipul metodei addObject:

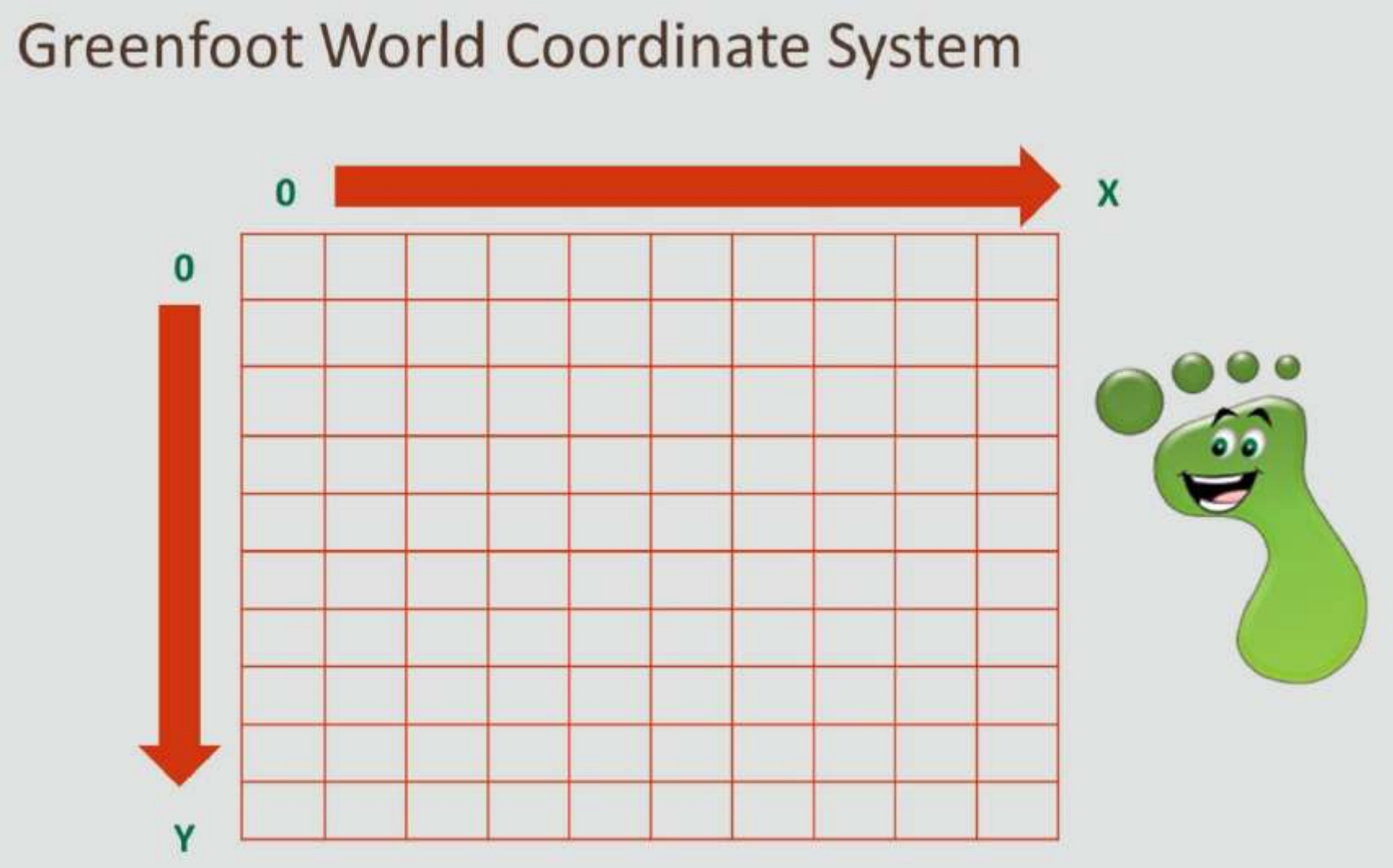
```
void addObject(Actor object, int x, int y)
```

Cuvantul cheie new

- acesta creeaza noi instante ale claselor deja existente
- procesul incepe cu (,) cuvantul cheie ‘new’, urmat de constructorul pe care doriti sa il apelati
- Lista cu parametrii ofera valori constructorului de care este nevoie pentru a initializa instata obiectului cu anumite variabile
- Constructorul standard are o lista de parametri vida si ofera instantei obiectului valorile standard.

```
new Constructor-name()
```

Sistemul de Coordonate ale super clasei World



Exemplu de adaugare de obiecte cu ajutorul metodei addObject

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class DukeWorld extends World
{

    /**
     * Constructor for objects of class DukeWorld.
     *
     */
    public DukeWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        addObject(new Duke(), 150,100);
    }
}
```

1. Utilizarea generarii numerelelor aleatoare (aranjarii intamplatoare) si intelegerea notatiei Dot Notation si a Constructorilor

2. Definirea metodelor

Plasarea eficienta a metodelor

Uneori, sunt necesare multe linii de cod sursa pentru a programa un anume comportament.
De exemplu, s-ar putea sa doriti sa programati un obiect sa manance alt obiect sau sa se invarta cand ajunge la capatul nivelului.
Definiti noi metode pentru a economisi timp si linii de cod:

- ✓ Definiti o noua metoda pentru a actiona de fiecare data cand este necesara o anume actiune
- ✓ Apelati metoda noua unde este necesara
- ✓ Definiti metoda in superclasa daca doriti ca subclasele sa primeasca automat aceasta metoda.

Definirea metodelor

Metodele definite sunt metode noi create de programator.
Aceste metode:

- pot fi executate imediat sau stocate si apelate uterior
- nu modifica comportamentul clasei cand sunt stocate
- separa codul in metode mai mici, facandu-l mai usor de citit.

Metodele definite creaza noi metode pe care clasa nu le are deja.
Aceste metode sunt scrise in codul sursa al clasei respective.

Pasii necesari pentru a defini o noua metoda

1. Selectati numele metodei

2. Deschideti editor-ul codului pentru clasa care va folosi metoda

3. Adaugati codul sursa pentru a defini metoda in metoda Act

4. Apelati aceasta noua metoda la inceputul metodei Act sau stocati-o si apelati-o mai tarziu

“Intoarcere de la Capatul Lumii”

Problema:

- ✓ Cand obiecte ajung la capatul lumii, atunci sunt incapabile de a se misca
- ✓ Obiectele in cauza ar trebui sa se intoarca si sa se miste in continuare in interiorul lumii

Solutie:

- ➢ Definiti o metoda in Superclasa Animal pentru a da tuturor subclaselor Animal abilitatea de a se intoarce si misca la capatul lumii.
- ➢ Apelati metoda noua in subclasele care ar trebui sa fie acum capabile de a se intoarce si misca

Testarea unui obiect care a ajuns la capatul lumii

Pentru a testa daca un obiect este aproape de capatul lumii, se testeaza utilizand:

- Multiple expresii logice pentru a indica daca una sau ambele conditii sunt adevarate sau false
- Operatori logici pentru a conecta expresiile logice folosite sa evalueze conditiile

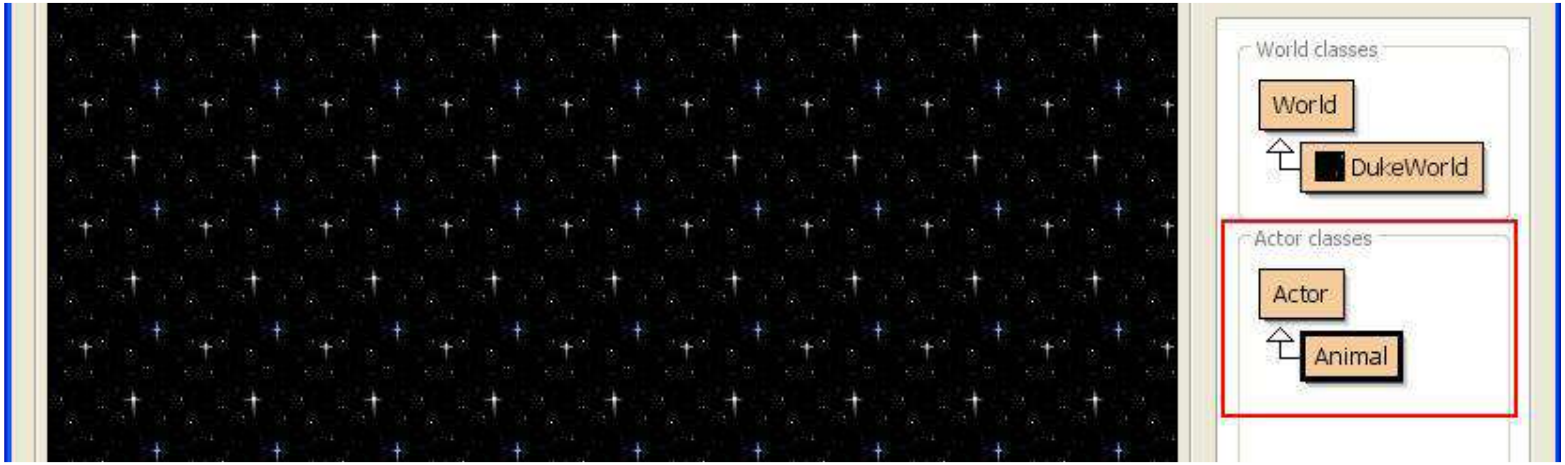
Operatorii logici

Operatorii logici pot fi folositi pentru a combina mai multe expresii logice intr-o singura expresie logica.

Logic Operator	Means	Definition
Exclamation Mark (!)	NOT	Reverses the value of a boolean expression (if b is true, !b is false. If b is false, !b is true).
Double ampersand (&&)	AND	Combines two boolean values, and returns a boolean value which is true if and only if both of its operands are true.
Two lines (II)	OR	Combines two boolean variables or expressions and returns a result that is true if either or both of its operands are true.

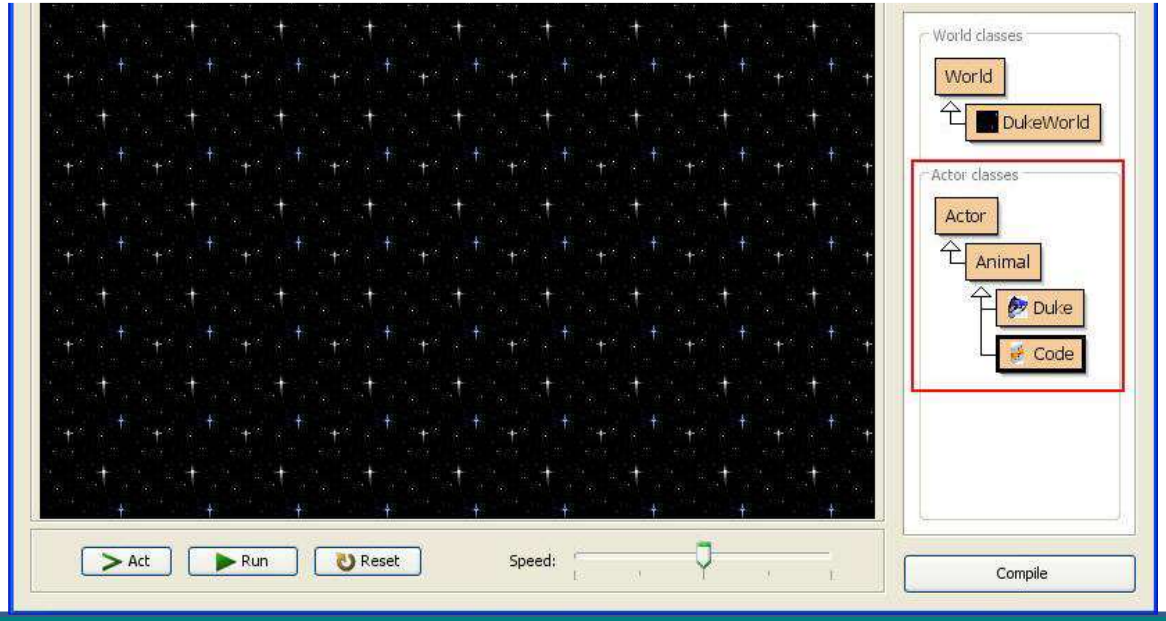
Crearea Superclasei Animal

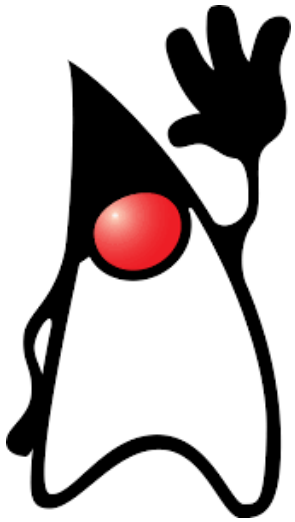
Inainte de crearea metodelor definite, trebuie creata o noua subclasa a clasei Actor pe care o puteti numi Animal.
Aceasta clasa nu are imagine si nu va avea niciun fel de integrare in scenariu, dar va stoca anumite metode definite pe care alte subclase le vor putea mosteni.



Crearea Superclasei Animal

- ✓ Creati subclase a superclasei Animal care vor avea instante ce ruleaza in scenariu.
- ✓ In acest exemplu, o subclasa Duke si subclasa Code au fost adaugate jocului, acolo unde Duke *(mascota Java)*, mananca obiectul Code.





Definirea Metodei atWorldEdge in Superclasa

- Deschideti editorul Code pentru superclasa Animal
- Scrieti codul pentru metoda atWorldEdge
- Compilati codul si apoi inchideti editorul

```
/**
 * Test if we are close to one of the edges of the world. Return true is we are.
 */
public boolean atWorldEdge()
{
    if(getX() < 20 || getX() > getWorld().getWidth() - 20)
        return true;
    if(getY() < 20 || getY() > getWorld().getHeight() - 20)
        return true;
    else
        return false;
}
```

Metode folosite in atWorldEdge

Metode folosite in atWorldEdge includ:

- getX: o metoda Actor care returneaza coordonata X a locatiei actuale a actorului
- getY – o metoda Actor care returneaza coordonata Y a locatiei actuale a actorului
- getWorld – o metoda Actor care returneaza lumea in care acest actor traieste
- getHeight – o metoda din clasa GreenfootImage care returneaza inaltimea imaginii.
- getWidth – o metoda a clasei GreenfootImage care returneaza latimea imaginii
- ||: - Simbol pentru o expresie conditionala SAU

Apelarea metodei atWorldEdge

Apelarea metodei atWorldEdge in subclasa

- Deschideti editorul codului pentru subclasa Animal
- Creati o structura IF care apeleaza metoda atWorldEdge ca pe o conditie. Conditia ghideaza cu cat trebuie obiectul sa se roteasca (in grade) daca conditia este adevarata
- Compilati codul si executati scenariul pentru testare

Apelarea metodei atWorldEdge

```
public class Bee extends Animal
{
    /**
     * Act - do whatever the Bee wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        if (atWorldEdge())
        {
            turn(19);
        }
        move(2);
    }
}
```

Documentatia Clasei

Documentatia clasei Animal arata metoda atWorldEdge dupa ce este definita.
Toate subclasele superclasei Animal dobandesc aceasta metoda.

Method Summary

void	**act**() Act - do whatever the Animal wants to do.
boolean	**atWorldEdge**() Test if object is close to one of the edges of the world.

Methods inherited from class

addedToWorld, getImage, getIntersectingObjects, getNeighbours, getObjectsAtOffset, getObjectsInRange, getOneIntersectingObject, getOneObjectAtOffset, getRotation, getWorld, getX, getY, intersects, move, setImage, setImage, setLocation, setRotation, turn

Methods inherited from class

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Metoda definita pentru “a manca obiecte”

- Puteti scrie cod in jocul dumneavoastra astfel incat un obiect pradator este capabil sa manance obiectele-prada.
- Creati o metoda definita in metoda Act din superclasa Animal care se numeste canSee pentru a permite obiectelor din subclasa Animal sa manance alte obiecte.
- Pentru a crea aceasta metoda definiti:
  - declarati o variabila pentru prada
  - folositi un operator de alocare pentru a stabili valoarea variabilei egala cu valoarea din metoda getOneObjectAtOffset

Definitia Metodei canSee

- Definiti metoda canSee in superclasa Animal.
- Aceasta metoda returneaza ‘adevarat’ daca obiectul pradator (Duke) aterizeaza pe obiectul prada (Code)

```
public class Animal extends Actor
{
    /**
     * Act - do whatever the Animal wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
    /**
     * Return true if Duke can see an object of the Code class 'clss'. Return false if there is no Code object here.
     */
    public boolean canSee(Class clss)
    {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        return actor != null;
    }
}
```

Metoda getOneObjectAtOffset

- Aceasta metoda returneaza un obiect care este localizat intr-o celula mentionata (apropiata de locatia obiectului)
- Revedeti aceasta metoda in documentatia clasei Actor

```
    /**
     * Return true if Duke can see an object of the Code class 'clss'. Return false if there is no Code object here.
     */
    public boolean canSee(Class clss)
    {
        Actor actor = getOneObjectAtOffset(0, 0, clss);
        return actor != null;
    }
}
```

Definitia metodei eat

- Creati o metoda definite in superclasa Animal numita eat(mananca).
- Aceasta metoda il va instrui pe Duke sa il manance pe Code daca aterizeaza pe el.

```
/**
 * Eat an object of class 'clss' if there is such an object in our current location.
 * Otherwise do nothing.
 */
public void eat(Class clss)
{
    Actor actor = getOneObjectAtOffset(0, 0, clss);
    if(actor != null) {
        getWorld().removeObject(actor);
    }
}
```

Definitia metodei lookForCode

- Creati o metoda definita in subclasa Animal (Clasa Duke) numita lookForCode care verifica daca un obiect Duke a aterizat pe un obiect Code.
- Daca da, obiectul Duke va manca obiectul Code.

```
public class Duke extends Animal
{
    /**
     * Act - do whatever the Duke wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
    /**
     * Look for code. If Duke finds code, he eats it. Otherwise, he does nothing.
     */
    public void lookForCode()
    {
        if (canSee(Code.class))
        {
            eat(Code.class);
        }
    }
}
```

Apelul metodei lookForCode

- Apelati noua metoda lookForCode in metoda Act a metodei Duke.
- Porniti animatia pentru a testa codul.

```
public class Duke extends Animal
{
    /**
     * Act - do whatever the Duke wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        move(3);
        if (Greenfoot.isKeyDown("left"))
        {
            turn(-2);
        }
        if (Greenfoot.isKeyDown("right"))
        {
            turn(2);
        }
        lookForCode();
    }
```

Întrebări?