Универзитет у Београду

Математички факултет

**МАСТЕР РАД**

# Примена образовног софтвера Greenfoot у настави Рачунарства и информатике у другом разреду средње школе

**Ментор:**

Проф. др Мирослав Марић

**Кандидат:**

Милена Вукићевић

Београд 2017.

# Садржај

# Увод

Образовни софтвер *Greenfoot* је један од најпопуларнијих образовних алата уз помоћ којег корисници могу да стекну искуство у објектно оријентисаном програмирању [1]. *Greenfoot* омогућава развој графичких апликација у *Javi*, једном од најраспрострањенијих програмских језика данашњице. Поред тога, *Greenfoot* би требало да подстакне алгоритамски начин размишљања код ученика који се први пут срећу са програмирањем.

Учење програмирања почетницима често може да представља проблем независно од узраста. Главни изазов представља изградња алгоритамског начина размишљања који је кључан за савладавање поступака и методологије решавања проблема. Додатну препреку у учењу представља крута синтакса програмског језика. Данас су у употреби многобројни програмски језици различитих намена и тешко је одабрати онај који је погодан за учење програмирања. Такав језик би требало да задовољи два битна услова:

- ✓ да буде једноставан и флексибилан (како би први програмски кораци били лаки);
- ✓ да буде добра подлога за наставак учења програмирања.

За учење програмирања се годинама уназад са великим успехом примењују образовни софтвери. Они су првенствено намењени као *инструмент* за учење, а мање као *алат* за писање програма за решавање неког проблема.

Неки од најпознатијих образовних софтвера су:

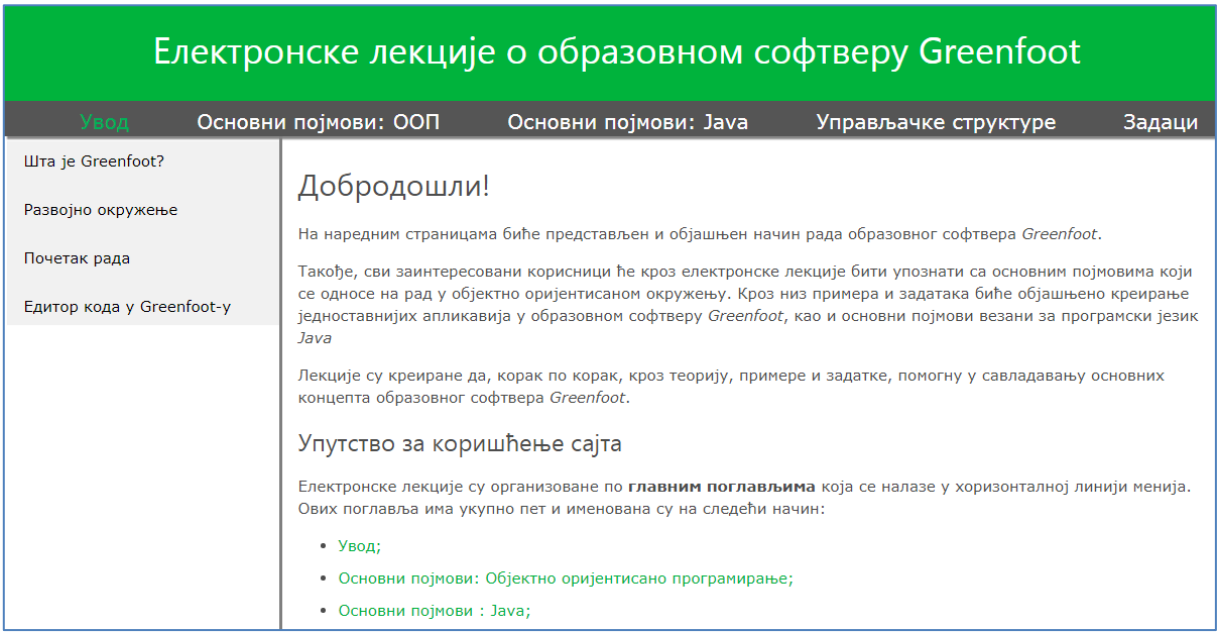
- ✓ ***Scratch*** – првенствено дизајниран за ниже разреде основне школе. Направљен је на *MIT* универзитету. О популарности овог софтвера говори и чињеница да се једно време користио за учење програмирања на Харварду;
- ✓ ***Alice*** – образовни софтвер намењен за више разреде основне школе;
- ✓ ***Greenfoot*** – намењен је за ученике средњих школа који имају, али и не морају да имају, неко предзнање из програмирања.

# 1. Електронске лекције о примени образовног софтвера Greenfoot

Електронске лекције о примени образовног софтвера *Greenfoot* у настави рачунарства и информатике у другом разреду средње школе доступне су на следећој адреси:

http://alas.matf.bg.ac.rs/~ml05206/master/index.html

Отварањем горе наведеног линка приказује се почетна страна електронских лекција. Изглед почетне странице сајта је приказан на слици 1.




**Слика 1. Почетна страна електронских лекција**

Електронске лекције су организоване по поглављима која се налазе у хоризонталној линији менија. Укупно има пет поглавлља именованих на следећи начин:

- ✓ Увод;
- ✓ Основни појмови: Објектно оријентисано програмирање;
- ✓ Основни појмови: *Java;*
- ✓ Управљачке структуре;
- ✓ Задаци.

Одабиром једног од понуђених поглавља отвара се помоћни мени који се налази са леве стране. Помоћни мени има карактеристичне ставке које су везане за одабрано поглавље. Корисници прво бирају главни, а затим помоћни мени, и на тај начин се приказује одговарајући садржај везан за ту електронску лекцију.

# 2. Образовни софтвер Greenfoot

Образовни софтвер *Greenfoot* је један од савремених софтвера за учење објектно оријентисаног програмирања. *Greenfoot* користи окружење програмског језика *Java* и подржава развој графичких апликација. Иако је осмишљен као помоћни образовни алат за учење програмирања у средњим школама, могу га користити и сви они који се први пут сусрећу са програмирањем. *Greenfoot* омогућава лак развој *2D* графичких апликација, као што су разне симулације и интерактивне игрице.

## 2.1. Историјат и верзије образовног софтвера Greenfoot

Образовни софтвер *Greenfoot* је осмишљен и развијен 2003. године на Кент универзитету (*University of Kent*) од стране Мајкл Колинга (*Michael Kölling*) и Пол Хенриксена (*Poul Henriksen*) уз подршку Оракл корпорације (*Oracle Corporation*) [2].

Прва верзија која је пуштена у употребу, *Greenfoot 1.0*, објављена је 31.5.2006. године. У мају 2007. године, *Greenfoot* је добио награду *Duke's Choice Award* у категорији „*Java* технологија у образовању“. У мају 2009. године *Greenfoot* постаје бесплатан и *open source* софтвер, да би пар месеци касније био представљен као званични алат наставницима широм света за учење програмирања. Убрзо је стекао широку примену као бесплатан софтвер за личну и некомерцијалну употребу који кроз интерактивни приступ развија алгоритамски начин размишљања.

Захваљујући програмском језику *Java,* као једном од најпопуларнијих објектно оријентисаних програма данашњице, *Greenfoot* представља добру базу за даље учење и олакшан прелазак на друге програмске језике.

# 3. Развојно окружење образовног софтвера Greenfoot

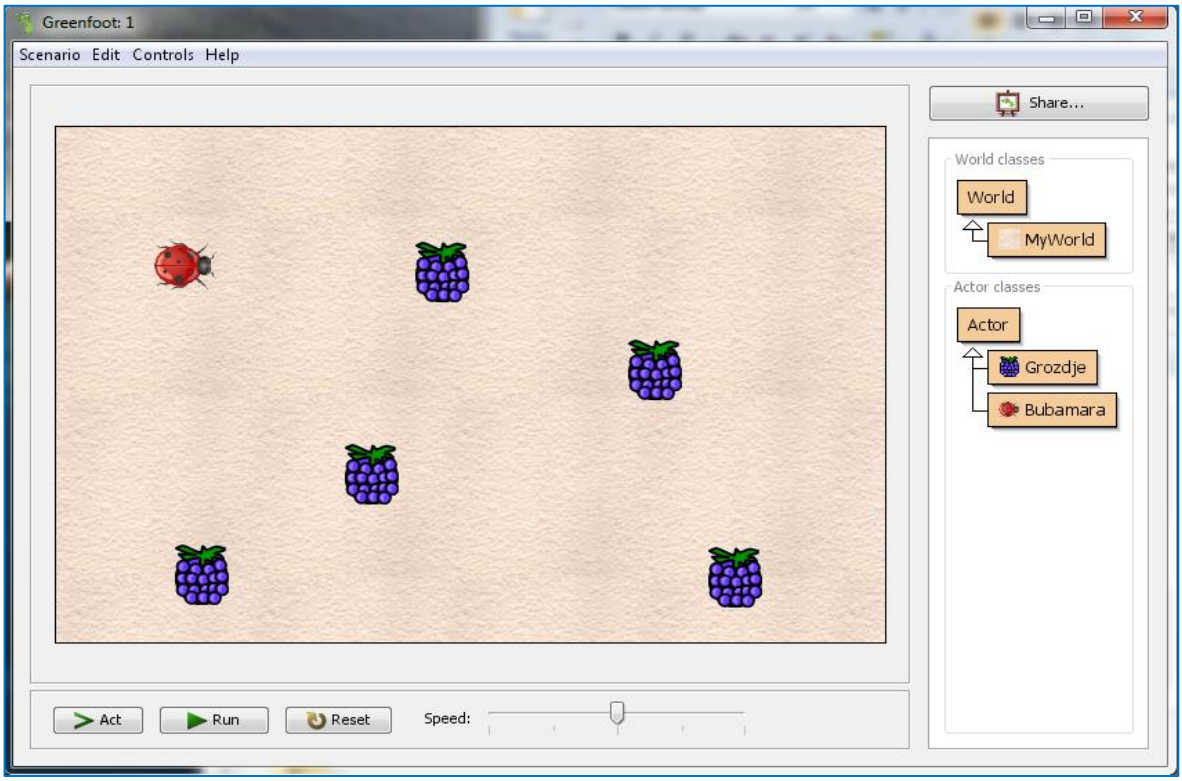
У овом поглављу представљени су елементи интегрисаног окружења образовног софтвера *Greenfoot* и начин на који се тим елементима управља. Затим је обрађен део који се односи на изглед главног прозора за креирање апликација. На крају поглавља дато је објашњење изгледа едитора за писање кода.

## 3.1. Елементи графичког корисничког интерфејса (GUI)

Графички кориснички интерфејс **(GUI – Graphical User Interface)** је врста интерфејса који омогућава корисницима да комуницирају са системом најчешће уз помоћ графичких иконица. Представља јако битан део окружења јер је једини видљив спољашњим корисницима.

Графички кориснички интерфејс користи визуелне елементе као што су: прозори, менији, дугмад, иконице, итд. које кориснику представљају додатне информације и омогућавају му потребне акције за манипулацију.

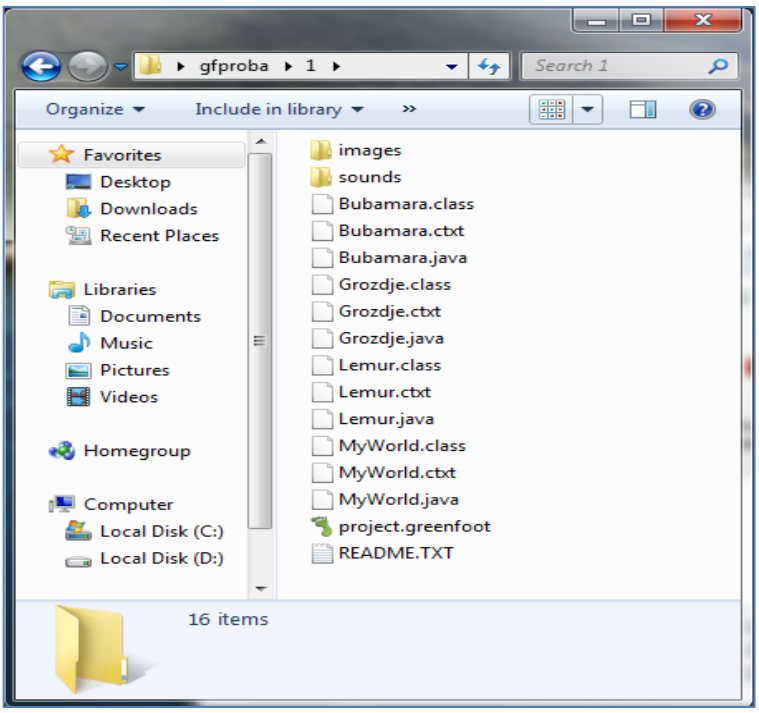
На слици 2 приказан је изглед интегрисаног окружења за развој апликација у едукативном софтверу *Greenfoot*.




Слика 2. Изглед развојног окружења

### 3.1.1. Сценарио

Радно окружење образовног софтвера *Greenfoot* је представљено у виду **сценарија** (*Scenario*). Сценарио заправо представља апликацију која ће бити извршена од стране корисника [3]. Сви фајлови који се генеришу приликом чувања и покретања образовног софтвера *Greenfoot* налазе се у посебном фолдеру креираном од стране корисника (слика 3).

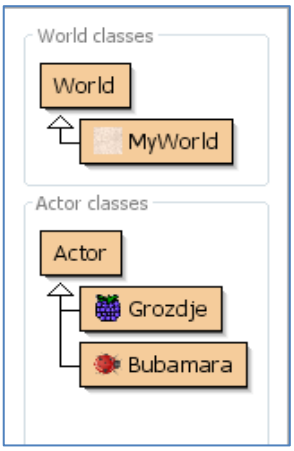



Слика 3. Пример сценарија са аутоматски генерисаним фајловима

Графички кориснички интерфејс сценарија је подељен на два главна сектора:

- ✓ **Свет** (*World*) – област у којој се извршава апликација. То је највећа област која се појављује приликом покретања образовног софтвера *Greenfoot* и која је предвиђена за тестирање и покретање апликација које корисник осмисли;
- ✓ **Дијаграм класа** – област у којој се креирају класе које се користе у оквиру апликације. Дијаграм класа је подељен у два сектора (слика 4):

1. ***World class*** – представља класу на основу које се креира област у којој се извршава апликација;
2. ***Actor class*** – садржи све класе које се користе у апликацији.




Слика 4. Пример дијаграма класа

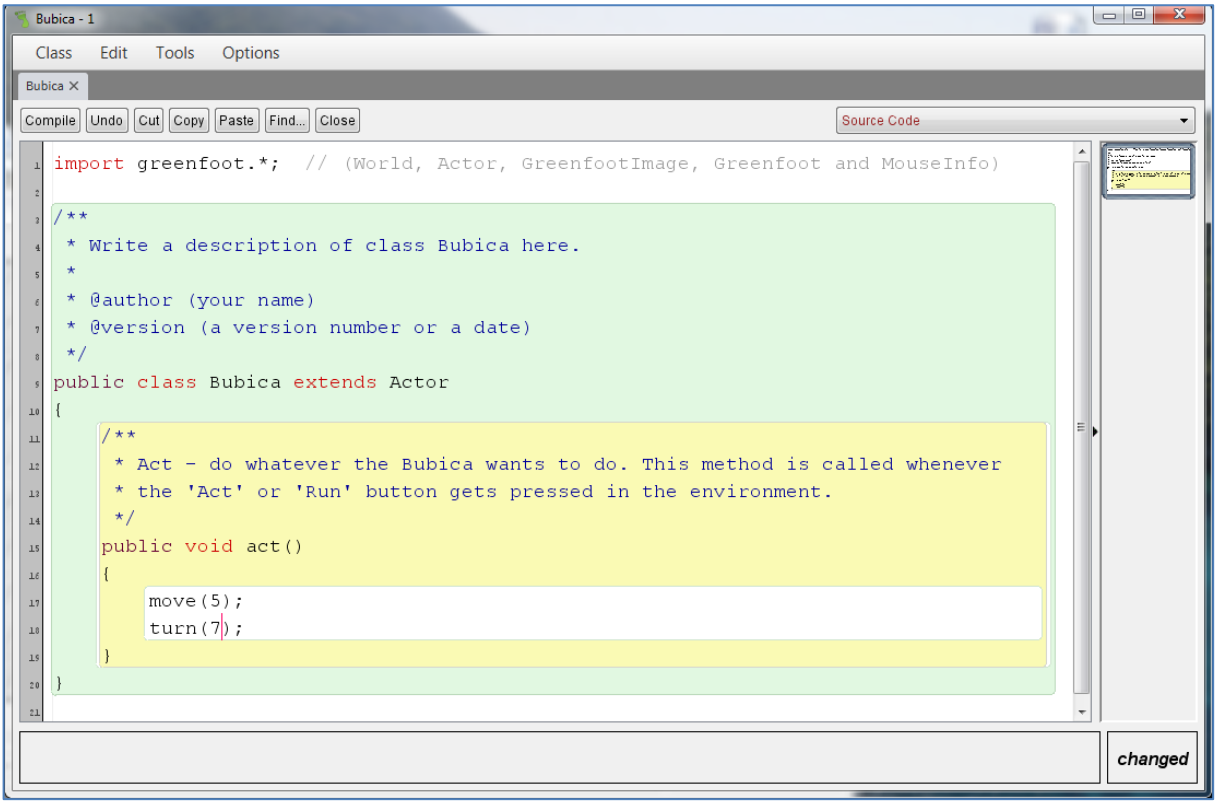
***Напомена:***

Појам класе је додатно објашњен у поглављу *4.1*.

### 3.1.2. Едитор за писање кода

Едитор за писање кода је део образовног софтвера *Greenfoot* који омогућава кориснику писање наредби у програмском језику *Java*. Едитор представља алат уз помоћ којег корисник дефинише шта креирани објекти раде у апликацији. У едитору се могу наћи сви елементи програмског језика *Java*, али за потребе ученика у средњим школама акценат је стављен на писање метода као и на манипулисање објектима.

Едитор је подељен у **блокове** који су означени са више различитих боја (слика 5):



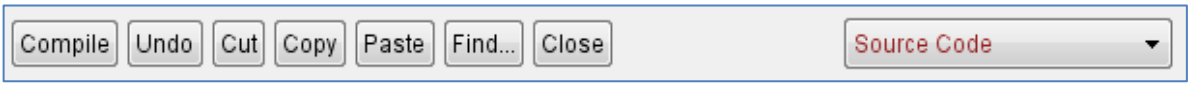

Слика 5. Пример едитора кода

- ✓ **Зелени блок** – представља класу коју је корисник креирао, а то је у нашем случају класа *Bubamara* (пример 1);
- ✓ **Жути блок** – представља све оно што желимо да се деси кликом на дугме *Act*. Унутар њега се налази метод *Act*;
- ✓ **Бели блок** – представља део кода који се односи на наредбе у програмском језику *Java* које корисник жели да напише;

- ✓ **Плави блок** – јавља се по потреби, уколико су као наредбе написане неке од управљачких структура у програмском језику *Java* (поглавље 8).

Са десне стране едитора налази се **слајдер** који кориснику олакшава кретање кроз исписане редове кода.

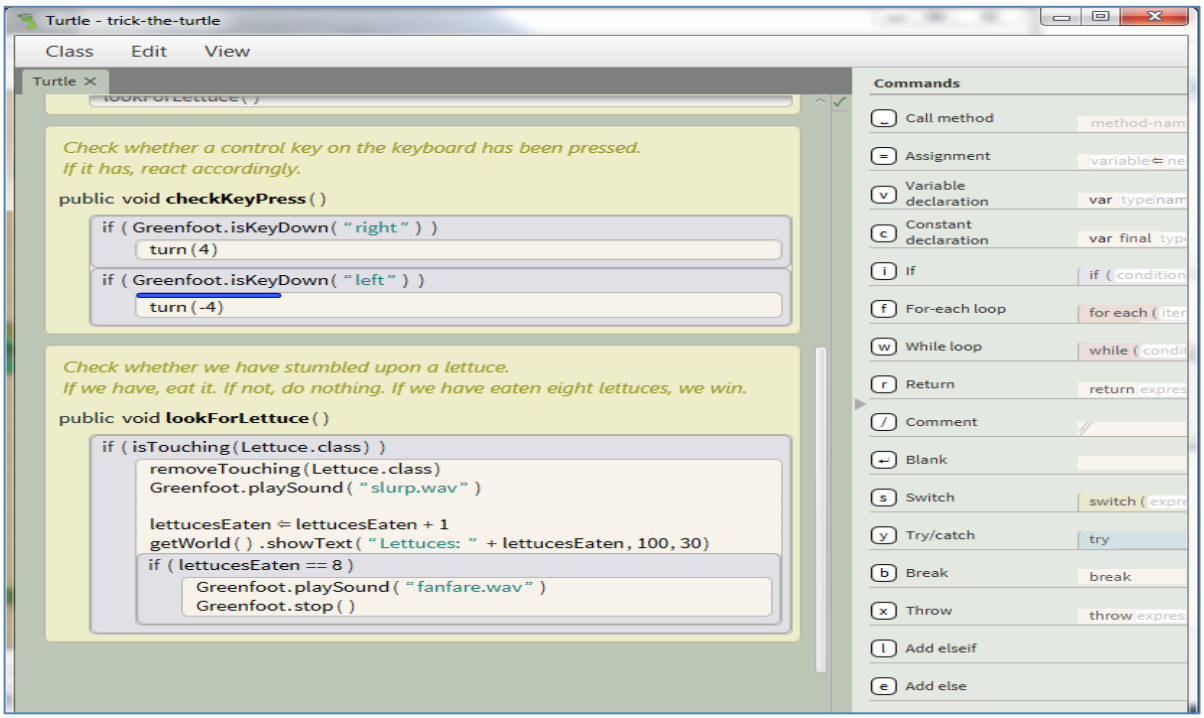
Едитор за писање кода садржи и **мени**је (*Class, Edit, Tools, Options*) који кориснику додатно омогућавају да прилагоди графички кориснички интерфејс својим потребама. Такође, у прозору едитора кода налазе се и додатна дугмад за манипулисање у развојном окружењу (слика 6).




Слика 6. Дугмад за манипулисање у развојном окружењу

Приликом покретања образовног софтвера *Greenfoot* појавиће се могућност да се сачува нови сценарио под окружењем програмског језика *Java* или под окружењем програмског језика *Stride*.

***Stride*** је програмски језик који је осмишљен од стране аутора образовног софтвера *Greenfoot*, Мајкл Колинга, са циљем да за ученике додатно олакша учење програмирања и појасни концепт објектно оријентисаног програмирања. Синтакса програмског језика *Stride* се разликује од синтаксе програмског језика *Java*. На слици 7 приказан је интегрисан едитор за програмски језик *Stride* са примером написаног кода.



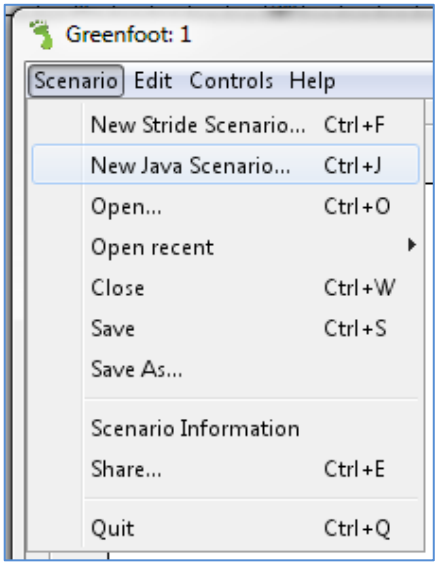

Слика 7. Едитор за писање кода у програмском језику *Stride*

**Напомена:**

За потребе овог мастер рада сви примери су урађени под окружењем програмског језика *Java*.

### 3.1.3. Додатни елементи графичког корисничког интерфејса

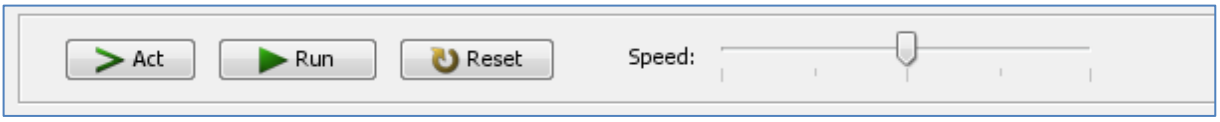
Као додатни елементи графичког корисничког интрфејса налазе се менији (*Scenario, Edit, Control, Help*) са додатним падајућим менијима. На слици 8 представљен је изглед менија са отвореним падајућим менијем *Scenario*.




Слика 8. Отворен мени *Scenario* са додатним опцијама

У окружењу су присутна и дугмад која служе за покретање, паузирање и ресетовање покренутог сценарија, као и слајдер којим може да се контролише брзина извршавања апликације (слика 9):

- ✓ **Act** – покретање апликације;
- ✓ **Run** – континуално извршавање апликације;
- ✓ **Reset** – поновно покретање апликације;
- ✓ **Speed** – слајдер којим се контролише брзина извршавања апликације.




Слика 9. Дугмад за контролу покренуте апликације

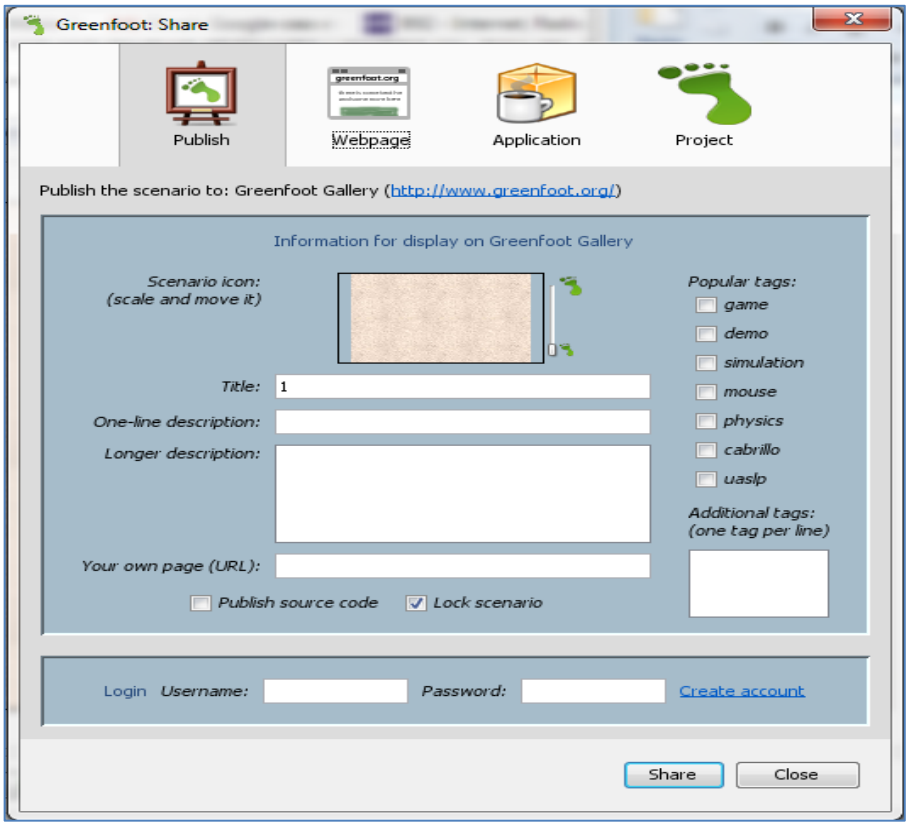
**Напомена:**

Разлика између дугмади *Act* и *Run* је у томе што дугме *Act* истовремено покреће једанпут све објекте који су постављени у област *Свет*, а *Run* непрекидно покреће апликацију.

Поред понуђених опција корисник има могућност да јавно објави своју апликацију на званичном сајту за *Greenfoot* (https://www.greenfoot.org/) кликом на дугме *Share* (слика 10).


Слика 10. Дугме за дељење апликације

Кликом на дугме *Share* отвара се нови прозор са четири додатне опције (*Publish, Webpage, Application, Project*) са додатним објашњењима за јавно објављивање и дељење апликације (слика 11).




Слика 11. Прозор за дељење и објављивање апликација

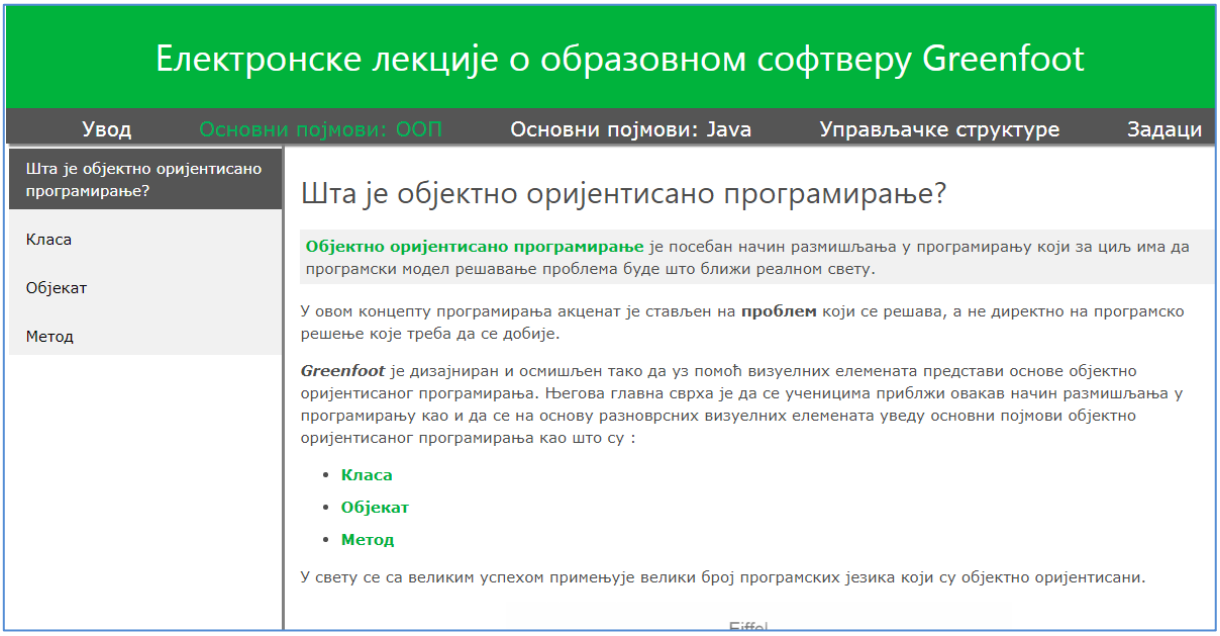
# 4. Објектно оријентисано програмирање

Објектно оријентисано програмирање је посебан начин размишљања у програмирању који за циљ има да програмски модел решавање проблема буде што ближи реалном свету. У објектно оријентисаном програмирању се много више времена троши на пројектовање апликације, а много мање на саму имплементацију (кодирање). У овом концепту програмирања акценат је стављен на проблем који се решава, а не директно на програмско решење које треба да се добије [4].

*Greenfoot* је дизајниран и осмишљен тако да експлицитно визуализује основне концепте објектно оријентисаног програмирања. Његова основна сврха је да се ученицима приближи овакав начин размишљања у програмирању као и да се на основу визуелних елемената уведу основни појмови објектно оријентисаног програмирања као што су:

- ✓ **Класа**;
- ✓ **Објекат**;
- ✓ **Метод**.

На примерима који следе објашњено је како *Greenfoot* ученицима на лак и разумљив начин уводи ове појмове. Поред тога, ови појмови су обрађени и у електронским лекцијама (слика 12).

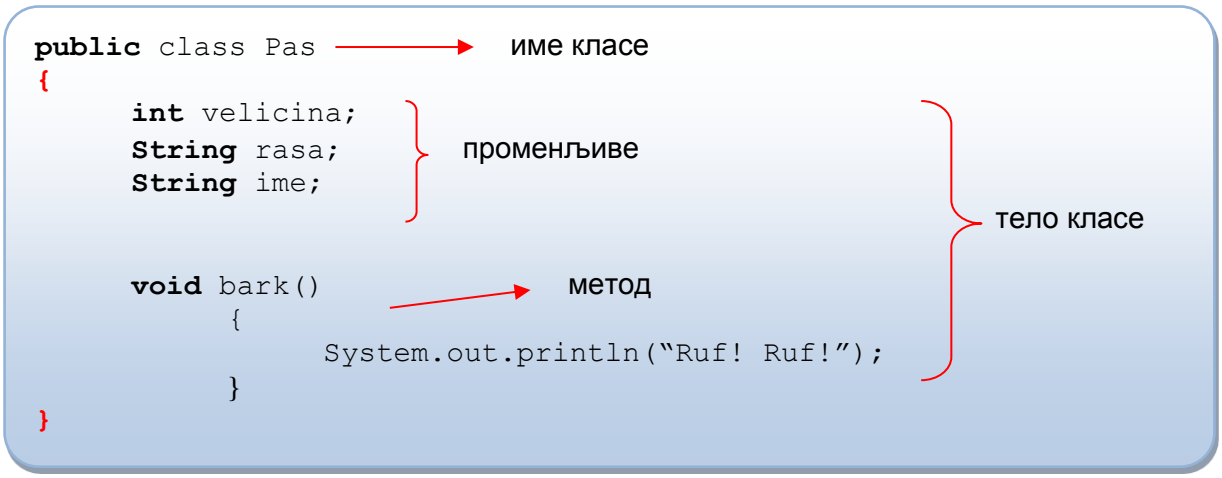



Слика 12. Електронска лекција – Објектно оријентисано програмирање

## 4.1. Класа

Класа је нацрт, односно шаблон, на основу кога ће се креирати готови објекти које ће корисник употребљавати у прављењу апликација. Класа је апстрактан појам који сам по себи нема практичну примену све док се из њега не направи објекат. Дакле, класе служе да би се из њих креирали објекти. Из једне класе се може креирати већи број објеката.

На слици 13 представљен је изглед једне класе у програмском језику *Java*.




Слика 13. Пример класе у програмском језику *Java*

### 4.1.1. Наслеђивање

Важно је напоменути да је у објектно оријентисаном програмирању јако битан појам **наслеђивања**. Програмски језик *Java* дозвољава да класа буде дефинисана и преко неке друге класе. Тако дефинисана класа се зове *потклаcа*, а родитељска класа *суперкласа*. Овакав начин креирања класа, којим се једна класа изводи из друге, назива се *наслеђивање*. У декларацији подкласе у програмском језику Java фигурише резервисана реч `extends` којом се означава која од класа је родитељска.

У образовном софтверу *Greenfoot* класа је аутоматски имплементирана у едитору за писање кода и има следећу општу структуру (слика 14):

- ✓ `import greenfoot.*` – симбол звездице означава да се цео пакет из софтвера *Greenfoot* увози у нашу апликацију тј. креирани сценарио. Ова опција је подразумевана приликом креирање нових класа у сценарију;
- ✓ **коментари** – додатно објашњавају шта класа представља и шта ради;
- ✓ `public class` – резервисане речи у програмском језику *Java* које указују да се ради о јавној класи која може да се користи било где у апликацији;
- ✓ `extends` – резервисана реч која објашњава да новокреирана класа наслеђује све методе од родитељске класе *Actor*;

- ✓ `methods` – омогућавају манипулисање објектима у апликацији. Методе се налазе у телу класе. Метод `act()` је подразумевани метод који се наслеђује из класе *Actor*.

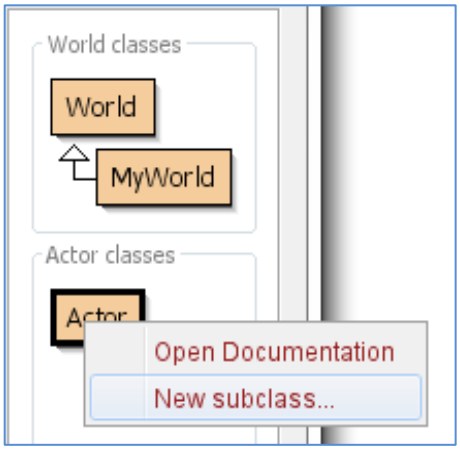
```
import greenfoot.*;

/**
 * A comment describing the class.
 */
public class CLASSNAME extends SUPERCLASS
{
    METHODS
}
```

Слика 14. Шаблон класе

***Пример 1***: Направити нову класу која се зове *Bubamara*.

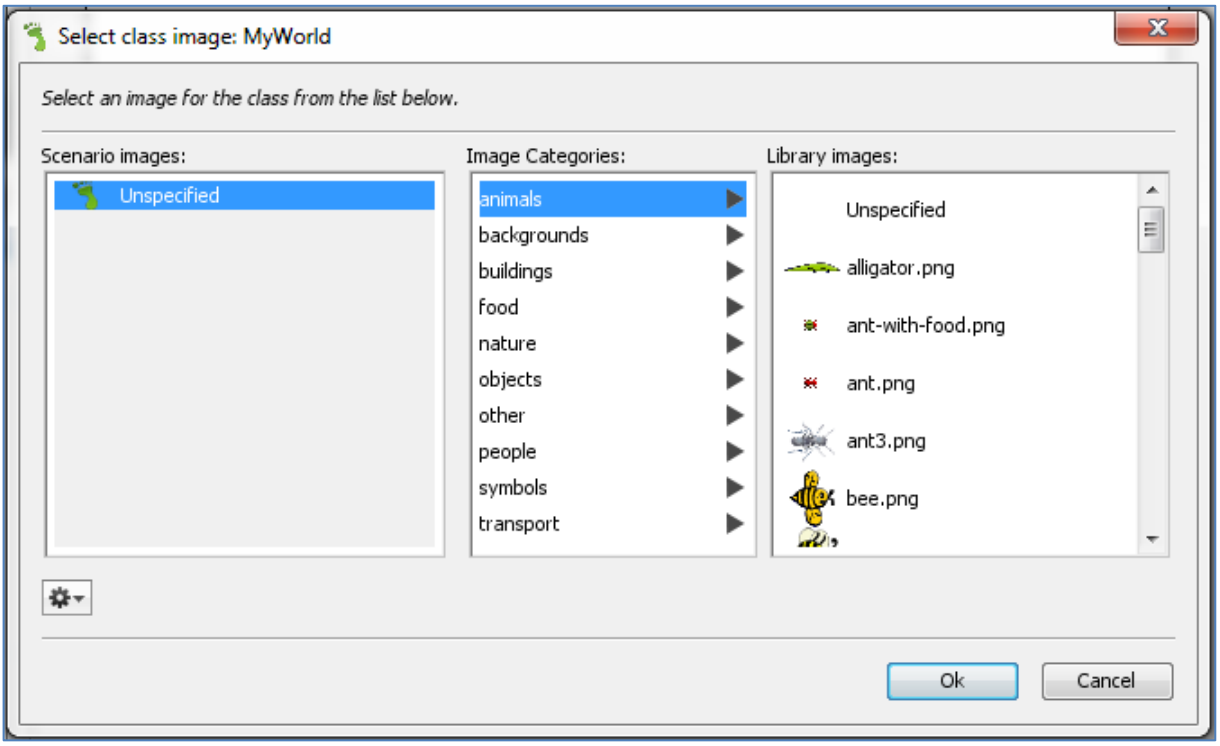
**Решење:**

Нова класа у образовном софтверу *Greenfoot* (која је потклаcа класе *Actor*) креира се тако што се најпре десним тастером миша кликне на класу *Actor* и изабере опција *new subclass* (слика 15).




Слика 15. Креирање класе

Након изабране опције појавиће се нови прозор где се додељује име класи и бира готова слика за изглед објеката који ће бити постављени у област *Свет* (слика 16). *Greenfoot* у понуди има велики избор графичких мотива који могу да се искористе као решења за изгед објеката. Такође, могуће је и поставити слику за изглед објекта која није у библиотеци слика.




Слика 16. Библиотека слика

**Напомена:**

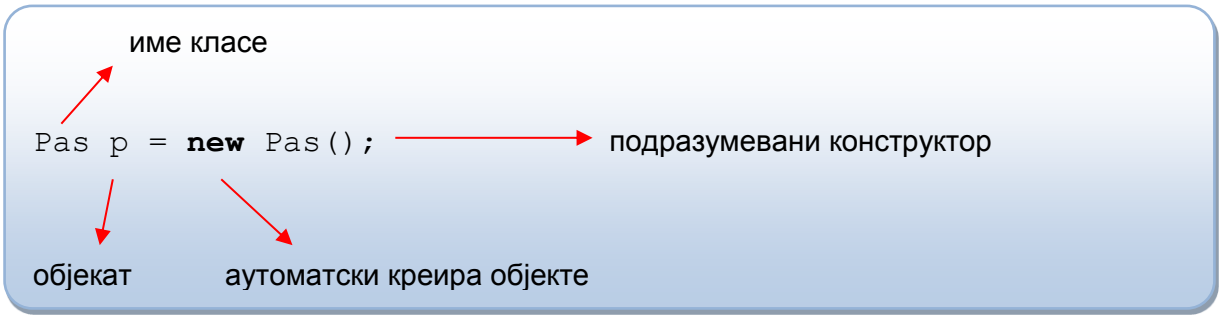
Додатна објашњења за класу *Actor* могу се наћи кликом на опцију *Open Documentation.* Коришћење документације у образовном софтверу *Greenfoot* додатно је објашњено у поглављу *4.3.1.*

## 4.2. Објекат

Појам објекта је један од кључних за разумевање принципа објектно оријентисаног програмирања. Објекат представља **инстанцу,** односно **примерак** класе који се креира на основу описа који се налазе у класи. Састоји се из:

- ✓ **Атрибута** – подразумевају све он што објекат јесте, тј. представљају својства тог објекта;
- ✓ **Метода** – подразумевају све оно што објекат може да ради, тј. методе су функције које служе за манипулисање објектима.

Објекти се у програмском језику *Java* декларишу на начин приказан на слици 17.

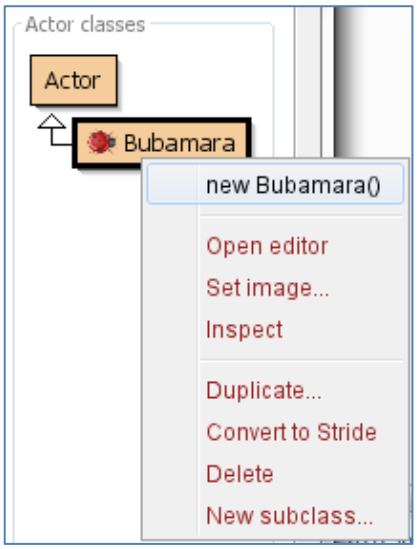



Слика 17. Декларисање објекта у програмском језиу *Java*

***Пример 2:*** *Из* постојеће класе *Bubamara* креирати нови објекат и поставити га у област *Свет* (*World*).

**Решење:**

У образовном софтверу *Greenfoot* из постојеће класе, која је названа *Bubamara,* нови објекат се креира тако што кликом на десни тастер миша изабере опција `new Bubamara()` (Слика 18). Нови објекат се затим поставља превлачењем мишем у област *Свет*, односно у област која је предвиђена за извршавање апликације. Уколико је потребно поставити више објеката у област *Свет*, неопходно је, држећи тастер *Shift,* кликнути на свако место у области *Свет* где се желе поставити нови објекти.

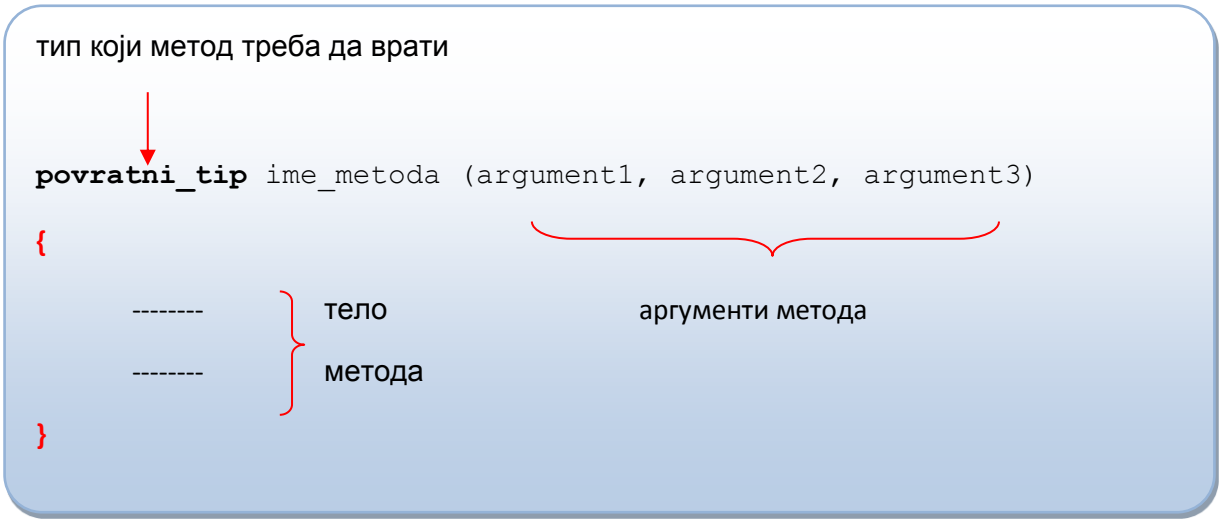



Слика 18. Креирање објекта из класе

## 4.3. Метод

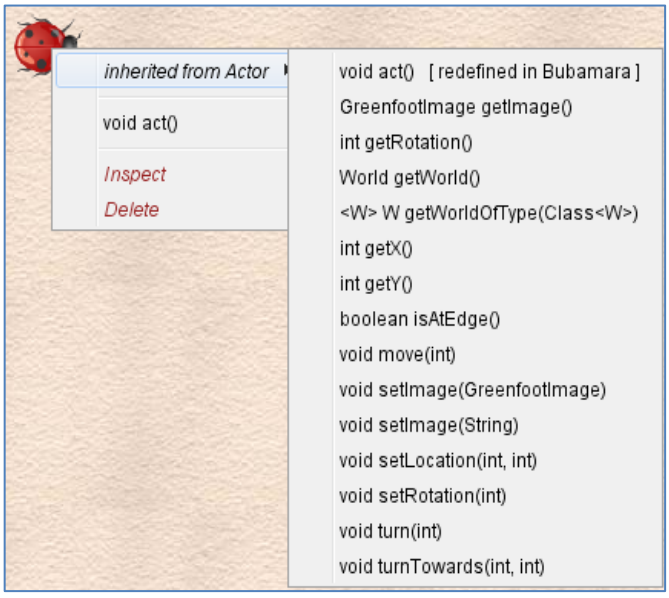
Метод у објектно оријентисаном програмирању представља „понашање“ објекта, односно, дефинише начин на који се може манипулисати објектом. Методе су једни од кључних појмова објектно оријентисаног програмирања, јер захваљујући њима објекти добијају улогу у апликацији коју креира корисник.

На слици 19 приказан је начин декларисања метода у програмском језику *Java*.




Слика 19. Декларисање метода у програмском језику *Java*

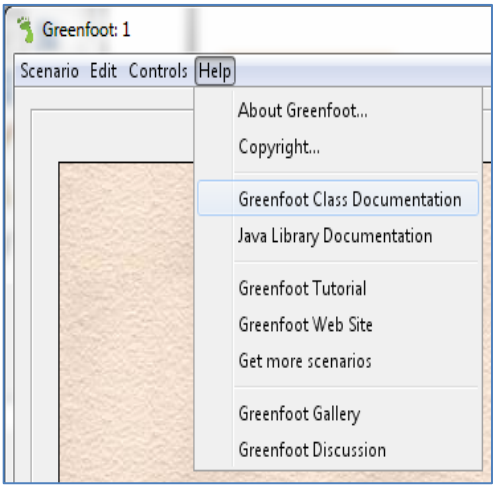
Свака постојећа класа у образовном софтверу *Greenfoot* има своју библиотеку метода које корисници могу да користе. Да би се проверило који метод може да се користи за одређену класу потребно је десним кликом на објекат изабрати опцију *inherited from Actor*. Након тога у каскадном менију појавиће се списак метода доступних за ту класу (слика 20).




Слика 20. Списак метода наслеђених из класе Actor

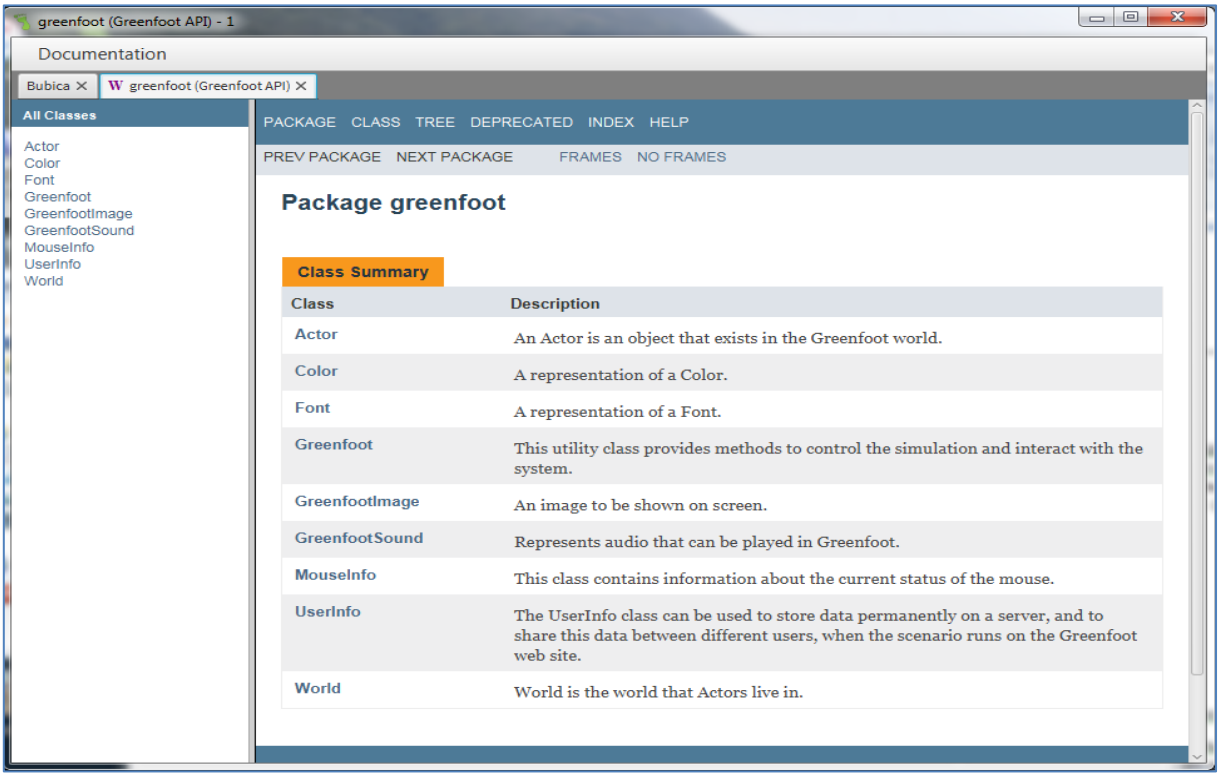
### 4.3.1. Документација

Уколико корисник није сигуран шта који метод представља и да ли је правилно синтаксно написан, образовни софтвер *Greenfoot* нуди могућност да се то провери уз помоћ **документације**. Документацији се приступа из менија *Help* избором опције *Greenfoot class documentation* (слика 21).




Слика 21. Приступ документацији

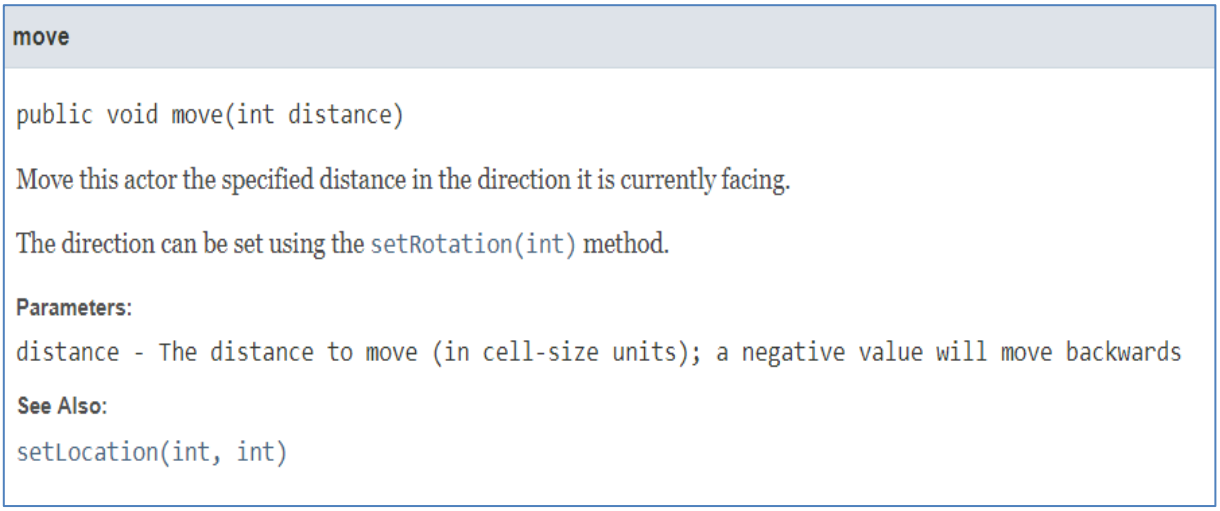
Након одабира ове опције појављује се прозор (слика 22) са детаљним објашњењима свих доступних класа и одговарајућих метода.




Слика 22. Прозор документације

Прозор документације састоји се из два сектора. Са леве стране се налази списак свих класа које сачињавају окружење *Greenfoot* (*Actor, Color, Font, Greenfoot, GreenfootImage, GreenfootSound, MouseInfo, UserInfo, World*). У области са десне стране налазе се сва потребна објашњења за сваку класу као и њене елементе.

Сваки метод има своју спецификацију која кориснику даје додатна објашњења за тај метод. На слици 23 представљена је спецификација метода `move()` који припада класи *Actor*.




Слика 23. Метод *move*

***Пример 3:*** Написати методе којима се објекат *Bubamara* из примера 2 кружно помера.

**Решење:**

У образовном софтверу *Greenfoot* метод представља део кода који се извршава кликом на дугме *Act*. Да би објекат који је постављен могао да реагује кликом на ово дугме, потребно је отворити едитор за писање кода двоструким кликом на класу *Bubamara* или десним кликом на тастер миша.

Након тога, потребно је у блоку метода `act()` написати два нова метода којима ће објекат *Bubamara* да се помера кружно (слика 24). Метод `move()` дефинише линеарно померање објекта за одређени број пиксела, док метод `turn()` дефинише угао под којим објекат скреће. У конкретном случају, објекат ће се померати за 5 пиксела удесно и скретати под углом од 7 степени.

**Напомена:**

Синтакса програмског језика *Java* налаже да се свака наредба одвоји интерпункцијским симболом тачка-зарез (**;**). Свака наредба се пише у блоку метода `act()` који је ограничен витичастим заградама.

```
public class Bubamara extends Actor
{
    /**
     * Act - do whatever the Bubica wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        move(5);
        turn(7);
    }
}
```

Слика 24. Решење задатка из примера 3

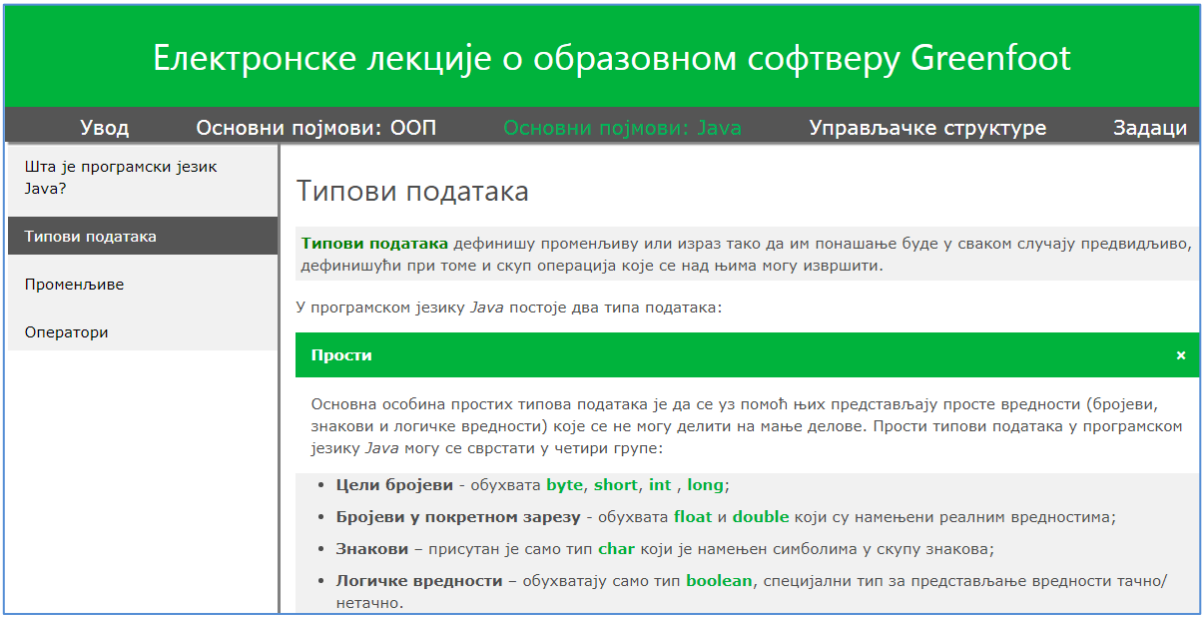
## 5. Типови података у програмском језику *Java*

Типови података дефинишу променљиву или израз тако да им понашање буде у сваком случају предвидљиво, дефинишући при томе и скуп операција које се над њима могу извршити. Свака променљива пре употребе мора бити **декларисана** (тј. мора јој се доделити тип), а пожељно је да буде и **иницијализована** (тј. да јој се додели вредност).

У програмском језику *Java* постоје два типа података:

- ✓ **Прости**;
- ✓ **Сложени**.

У електронским лекцијама дата је класификација типова података (слика 25).




Слика 25. Електронске лекције-Типови података

## 5.1. Прости типови података

Основна особина простих типова података је да се уз помоћ њих представљају просте вредности (бројеви, знакови и логичке вредности) које се не могу делити на мање делове. У програмском језику *Java* дефинисано је осам простих типова података, који се разликују према опсегу могућих вредности и величини меморијског простора који заузимају. Прости типови података могу се сврстати у четири групе:

- ✓ **Цели бројеви** – обухвата `byte`, `short`, `int` и `long`;
- ✓ **Бројеви у покретном зарезу** – обухвата `float` и `double` који су намењени реалним вредностима;
- ✓ **Знакови** – присутан је само тип `char` који је намењен симболима у скупу знакова;
- ✓ **Логичке вредности** – обухватају само тип `boolean`, специјални тип за представљање вредности тачно/нетачно (*true/false*).

За учење програмирања у образовном софтверу *Greenfoot* за потребе ученика другог разреда средње школе неопходни су најчешће прости типови података. У табели 1 за сваки прост тип података дата је његова подразумевана вредност, опсег вредности као и величина коју заузима у меморији.

Табела 1: Прости типови података

| Тип података | Подразумевана вредност | Опсег вредности | Подразумевана величина |
|---|---|---|---|
| **Boolean** | false | true или false | 1 bit |
| **Char** | '\u0000' | од '\u0000' до '\uffff' | 2 byte |
| **Byte** | 0 | од -128 до 127 | 1 byte |
| **Short** | 0 | од -32768 до 32767 | 2 byte |
| **Int** | 0 | од $-2^{31}$ до $2^{31}-1$ | 4 byte |
| **Long** | 0l | од $-2^{63}$ до $2^{63}-1$ | 8 byte |
| **Float** | 0.0f | од ±1.4E-45 до ±3.4028235E+38 | 4 byte |
| **Double** | 0.0d | од ±4.9E-324 до ±1.7976931348623157E+308 | 8 byte |

## 5.2. Сложени типови података

У сложене типове података спадају:

- ✓ **Низови**;
- ✓ **Стрингови**.

### 5.2.1. Низови

У програмирању се често јавља потреба да се за промењљиве припреми неколико десетина, па и стотине података. Тада би декларисање, а затим и манипулисање над, на пример, стотином променљивих било врло компликовано. У таквим случајевима користе се **индексиране променљиве**, односно, **низови**. Неке од предности коришћења низова су:

- ✓ ефикасан унос / испис елемената;
- ✓ елементи низа се не декларишу појединачно, већ истовремено;
- ✓ једноставније манипулисање подацима (сортирање, претраживање...).

У програмском језику *Java* могуће је креирати низ било ког типа података, па чак и низ објеката. Елементи низа се индексирају почев од нуле. То значи да почетни члан низа има индекс 0, други 1, итд.

У програмском језику *Java* низ се декларише на један од следећих начина:

```
tip [] ime_niza;
```

или

```
tip ime_niza[];
```

Приликом декларације низа, елементима низа се могу придружити иницијалне вредности. То се ради на следећи начин:

```
tip []ime_niza = {vrednost_1,vrednost_2, …, vrednost_n};
```

### 5.2.2. Стрингови

Тип података `String` је сложени тип података који може садржати више знакова, за разлику од једноставног типа података `char` који може садржати само један знак. Променљиве типа `char` се иницијализују коришћењем једноструких наводника (' '), док се променљиве типа `String` иницијализују коришћењем двоструких наводника (" ").

Тип података `String` представља једну класу у програмском језику *Java* и декларишу се на следећи начин:

```
String ime_promenljive;
```

Истовремена декларација и иницијализација промењиве типа `String`, врши се на следећи начин:

```
String ime_promenljive = "vrednost";
```

## 6. Промењиве

Промењиве у програмском језику *Java* служе за чување података. Могу се чувати нумеричке вредности, стрингови или објекти неке класе. Програмски језик *Java* је строго типизиран, па свака промењива има свој тип, име али не и обавезно вредност коју чува. У примерима који следе представљени су начини декларисања промењивих у програмском језику *Java*.

***Пример 4:*** Декларисање промењивих

```
int celobrojna_vrednost;

char karakter;

boolean jeste_tacno;
```

У примеру 5 приказано је како се иницијализују, односно додељују вредности промењивим.

***Пример 5:*** Иницијализација промењивих

```
celobrojna_vrednost = 7;

karakter = 'y';

jeste_tacno = true;
```

На примеру 6 приказано је како се промењива може иницијализовати у истом тренутку у ком се декларише.

***Пример 6:*** Декларисање и иницијализовање променљивих

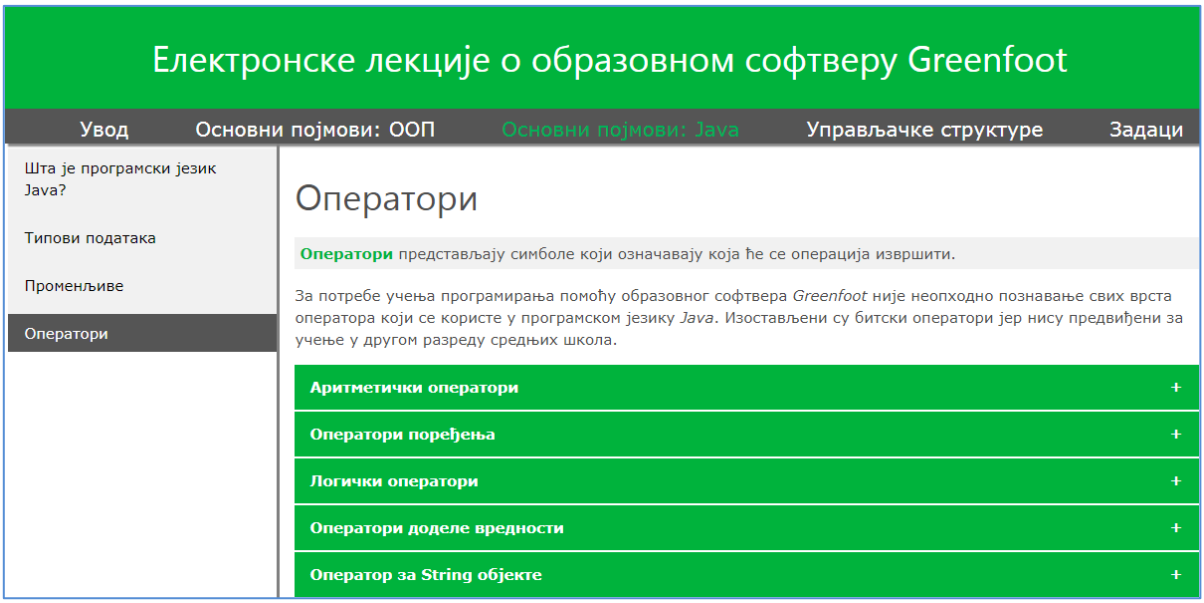
```
int celobrojna_vrednost = 7;

char karakter = 'y';

boolean jeste_tacno = true;
```

***Напомена:***

Програмски језик *Java* је *case sensitive*, односно разликује мала и велика слова. На пример, променљиве **`jeste_tacno`** и **`Jeste_tacno`** су две различите променљиве. Образовни софтвер *Greenfoot* се такође строго придржава овог принципа. У случају када је потребно, на пример, позвати неки метод из библиотеке, мора се строго водити рачуна како је написано његово име.

## 7. Оператори

Оператор представља симбол који указује на извршавање одређене математичке или логичке операције. У поглављима која следе представљени су најчешће коришћени оператори. Изостављени су битски оператори јер нису предвиђени за учење у другом разреду средњих школа. Класификација и примена одређених оператора обрађена је и у електронским лекцијама (слика 26).




Слика 26. Електронске лекције – Оператори

## 7.1. Аритметички оператори

Аритметички оператори служе за извршавање аритметичких операција са променљивама бројевног типа и могу се користити на вредностима било које нумеричке врсте (*byte, short, int, long, float,* или *double*). Уколико се врсте променљивих које учествују у изразу разликују, долази до превођења променљивих ако је то могуће. На пример, ако у изразу учествује једна променљива типа `int` и једна променљива типа `double`, променљива типа `int` биће преведена у тип `double`.

У табели 2 дат је преглед аритметичких оператора. Треба напоменути да уколико се примени оператор дељења над целобројним типом, врши се целобројно дељење.

Табела 2: Аритметички оператори

| Оператори | Опис |
|---|---|
| + | Сабирање |
| - | Одузимање |
| * | Множење |
| / | Дељење |
| % | Дељење по модулу |
| ++ | Инкрементирање (унарни оператор) |
| -- | Декрементирање (унарни оператор) |

## 7.2. Оператори поређења

Програмски језик *Java* подржава логичке (*boolean*) променљиве и изразе за описивање услова који могу бити тачни (*true*) или нетачни (*false*). Један од начина стварања логичког израза је упоређивање две вредности помоћу оператора поређења.

Оператори поређења испитују да ли су две вредности једнаке, различите, да ли је једна већа од друге, итд. У табели 3, представљени су симболи за операторе поређења као и њихов опис.

Табела 3: Оператори поређења

| Оператори | Опис |
|---|---|
| == | Једнакост |
| != | Неједнакост |
| < | Мање од |
| > | Веће од |
| <= | Мање или једнако |
| >= | Веће или једнако |

## 7.3. Логички оператори

Логички оператори се примењују над логичким вредностима и као резултат такође враћају логичку вредност. У табели 4 дат је преглед логичких оператора. Треба напоменути да су логичко И (&&) и ИЛИ (||) оператори који се примењују над два операнда, док се оператор негације (!) примењује над једним операндом.

Табела 4: Логички оператори

| Оператори | Опис |
|---|---|
| && | Логичко И |
| \|\| | Логичко укључујуће ИЛИ |
| ^ | Логичко искључујуће ИЛИ |
| ! | Негација |

## 7.4. Оператори доделе вредности

Оператори доделе вредности додељују променљивој са леве стране вредност израза са десне стране оператора. У табели 5 дат је детаљан опис оператора доделе вредности.

Табела 5: Оператори доделе вредности

| Оператор | Опис |
|---|---|
| = | Додела вредности. |
| += | Додела вредности са применом аритметичке операције сабирања. Запис a+=b одговара запису a=a+b. |
| -= | Додела вредности са применом аритметичке операције одузимања. Запис a-=b одговара запису a=a-b. |
| *= | Додела вредности са применом аритметичке операције множења. Запис a*=b одговара запису a=a*b. |
| /= | Додела вредности са применом аритметичке операције дељења. Запис a/=b одговара запису a=a/b. |
| %= | Додела вредности са применом аритметичке операције дељења по модулу. Запис a%=b одговара запису a=a%b. |

## 7.5. Оператор за String објекте

Оператор **+**, који је већ описан у поглављу *7.1*, може да се и користи за надовезивање стрингова. У образовном софтверу *Greenfoot* стрингови се углавном користе приликом исписивања порука у току самог извршавања апликације.

### 7.6. Приоритет оператора

Ако се у једном изразу појави више оператора, поставља се питање којим редоследом ће они бити примењивани тј. извршавани. То је дефинисано приоритетом оператора. Највиши приоритет имају заграде, а приоритет осталих оператора дат је у табели 6.

Табела 6: Листа оператора према приоритетима

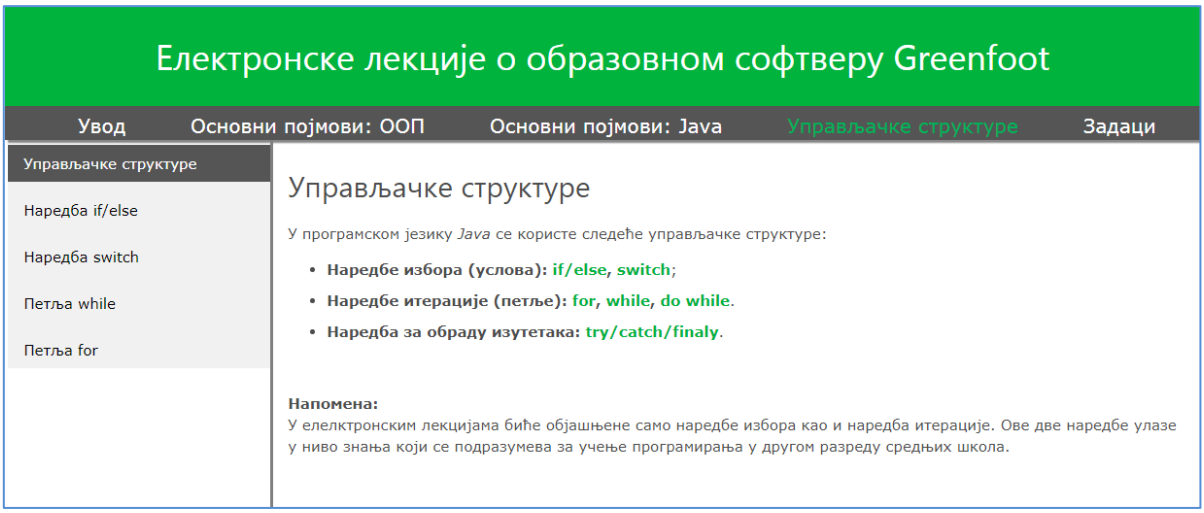
| | |
|---|---|
| Унарни оператори | **++, - -, унарни + и -** |
| Множење и дељење | ***, /, %** |
| Сабирање и одузимање | **+, -** |
| Оператори поређења | **>, <, >=, <=** |
| Једнакост и неједнакост | **==, !=** |
| Логичко И | **&&** |
| Логичко ИЛИ | **\|\|** |
| Условни оператор | **? :** |
| Оператори доделе | **=, +=, -=, *=, /=, %=** |

## 8. Управљачке наредбе

У програмском језику *Java* се користе следеће управљачке структуре:

- ✓ **Наредбе избора (услова)**: `if/else, switch`;
- ✓ **Наредбе итерације (петље)**: `for, while, do while`;
- ✓ **Наредба за обраду изутетака**: `try/catch/finally`.

Овде ће бите објашњене само наредба избора као и наредба итерације. Ове две класе наредби улазе у ниво знања који се подразумева за учење програмирања у другом разреду средњих школа. Изостављене су наредбе за обраду изутетака јер нису предвиђене за учење у другом разреду средњих школа. Све то је доступно и у електронским лекцијама (слика 27).




Слика 27. Електронске лекције – Управљачке структуре

## 8.1. Наредба if/else

Наредба `if/else` служи да се изврши један део програма (тј. једна или више наредби) у зависности од тога да ли је испуњен одређени услов или није. Представља једну од кључних наредби и неизоставну конструкцију у апликацијама образовног софтвера *Greenfoot*. Већина апликација које корисник буде направио у *Greenfoot* односи се на испитивање одређеног услова, како би се задата наредба (или више њих) извршила. Синтакса наредбе `if/else` има следећи облик у програмском језику *Java*:

```
if (uslov){
      uslovne_naredbe_1;
}
else {
      uslovne_naredbe_2; //grana nije obavezna
}
```

Компоненте наредбе `if/else` су:

- ✓ `uslov` – може бити тачан или нетачан;
- ✓ `uslovne_naredbe_1` – наредбе које ће се извршити једино ако је услов тачан;
- ✓ `uslovne_naredbe_2` – наредбе које ће се извршити ако је услов нетачан.

На примерима који следе, објашњено је како се користи наредба `if/else` у образовном софтверу *Greenfoot*.

***Пример 7:*** Направити метод *act()* којим се објекат *Bubamara* из примера 3 приликом клика на тастер *Left* или *Right* на тастатури скреће у области *Свет* под углом од 7 степени улево или удесно.

**Решење:**

У овом примеру неопходно је користити наредбу `if/else`. У услову за наредбу `if` користимо метод који у зависности да ли је одређени тастер притиснут или не враћа вредност тачно или нетачно *(true/false)*. У конкретном случају коришћен је метод `isKeyDown(String).` Овај метод за аргумент узима тип података `String` те је стога неопходно користити резервисане речи које се односе на тастере на тастатури за померање и кретање (у овом случају *left* и *right)*. Решење овог задатка илустровано је на слици 27.

```
public void act()
{
    move(5);
    if (Greenfoot.isKeyDown("left"))
    {
        turn(-7);
    }
    if (Greenfoot.isKeyDown("right"))
    {
        turn(7);
    }
}
```

Слика 27. Решење задатка из примера 7

***Напомена:***

Метод `isKeyDown(String)` се не налази у библиотеци класе *Actor*, већ је интегрисан као саставни део класе *Greenfoot*.

***Пример 8 (наставак примера 7):*** Направити нову класу *Grozdje* која је поткласа класе *Actor*. У област *Свет* поставити 7 нових објеката *Grozdje* и сачувати постојећу поставку апликације. Након тога креирати метод *act()* којим се објекат *Grozdje* уклања из области *Свет,* уколико га додирне објекат *Bubamara*.

**Решење:**

У примеру 2 објашњено је креирање нове класе, постављања нових објеката у област *Свет* као и чување постојеће апликације. Како би се решио следећи део задатка потребно је применити метод, `isTouching(_cls_)`, којим ће се проверити да ли је постојећи објекат додирнут или не. Уколико је овај услов тачан, примењује се метод, `removeTouching(_cls_)`, којим се постојећи објекат уклања из области *Свет*. На слици 28 налази се део кода који ће се извршити кликом на дугме *Act*.

```
if (isTouching(Grozdje.class))
{
    removeTouching(Grozdje.class);
}
```

Слика 28. Решење задатка из примера 8

***Пример 9 (наставак примера 8):*** Направити нову класу *Lemur* која је потклаcа класе *Actor*. У област *Свет* поставити поред постојећих објекта још 3 нова објекта *Lemur*. Након тога креирати метод којим се објекти *Lemur* крећу насумично по области *Свет*. Уколико ударе у ивицу области *Свет* потребно је да се окрену за 7 степени.

**Решење:**

За решење овог задатка неопходно је коритсити метод `getRandomNumber(int limit)` који је имплементиран у библиотеци класе *Greenfoot*. Метод `getRandomNumber(int limit)` враћа насумичан ненегативан број, строго мањи од лимита који корисник зада преко променљиве `limit`. На пример, ако корисник постави да је `limit` 15, метод враћа насумичан цео број из интервала [0, 14].

Један од начина да се објекти *Lemur* крећу насумично је да скрену под одређеним углом сваки пут када метод `getRandomNumber(int limit)` врати број мањи од неког унапред задатог броја. На пример, може се задати да лемури скрећу под углом од 7 степени сваки пут кад је генерисани број мањи од 10 (видети слику 29). Тиме је први део задатка решен.

За решење другог дела задатка потребно је користити метод `isAtEdge()` који враћа вредност тачно ако објекат удари у ивицу, а у супротном враћа вредност нетачно. Решење задатка дато је на слици 29.

```
public void act()
{
    move(5);
    if (Greenfoot.getRandomNumber(100)<10)
    {
        turn(7);
    }
    if (isAtEdge())
    {
        turn(7);
    }
}
```

Слика 29. Решење задатка из примера 9

***Пример 10 (наставак примера 9):*** Направити метод којим се објекат *Bubamara* уклања из области *Свет* уколико га додирне објекат *Lemur*. Уколико више не постоји објекат *Bubamara* исписати поруку „Изгубили сте!“

**Решење:**

Решење првог дела задатка је већ објашњено у примеру 8. Да би се решио део задатка који се односи на исписивање поруке, потребно је позвати метод `showText(_text_, _x_, _y_)` за приказивање текста у области *Свет*. Овај метод има три аргумента. Први аргумент представља текст у облику стринга који ће бити исписан, док друга два аргумента представљају координате у области за извршавање апликације где ће бити исписана порука. Треба истаћи да се метод `showText()` позива из класе `getWorld()`. Решење задатка приказано је на слици 30.

```
if (isTouching(Bubamara.class))
{
    removeTouching(Bubamara.class);
    getWorld().showText("Izgubili ste!", 300, 300);
}
```

Слика 30. Решење задатка из примера 10

## 8.2. Наредба switch

Наредба `switch` се користи код вишеструког гранања. Некада је пожељно да се избегне понављање вишеструких `if /else` наредби, те се у том случају користи ова наредба. Наредба `switch` омогућава процену услова и на основу те вредности изврши се „скок“ на одређено место унутар израза `switch`.

У образовном софтверу Greenfoot вредност израза који се процењује мора бити целобројног типа (*byte*, *short*, *int* или *char)* или `String`. Могуће је, такође, употребити резервисану реч `default` која је задужена за „скок“ унутар израза у случају да вредност израза не одговара ни једној од ознака случаја. Наредба `switch` има следећи облик:

```
switch ( promenljiva ) {
case (vrednost1) :
uslovne_naredbe
break ;
case (vrednost2) :
uslovne_naredbe
break ;
. . .
default :
uslovne_naredbe
}
```

Кључне речи ове наредбе су:

- ✓ `switch` – идентификује променљиву на основу које програм одлучује где ће даље да крене;
- ✓ `case` – означава почетак сваке гране;
- ✓ `break` – обележава крај сваке гране;
- ✓ `default` – путања којом ће програм кренути уколико променљива наредбе *switch* не одговара ниједној од вредности *case*.

***Пример 11:*** Направити апликацију којом се објекат *Bubamara* помера по области *Свет* у зависности од притиска на тастере: горе (објекат се окреће за 30 степени), доле (објекат се окреће за -30 степени), лево (објекат се помера за 7 пиксела улево), десно (објекат се помера за 7 пиксела удесно). Задатак решити применом наредбе `switch`.

**Решење:**

У решењу овог примера неопходно је да се декларише променљива `myKey`, типа `String`, која служи за чување имена притиснутог тастера. Име притиснутог тастера добија се позивом методе `Greenfoot.getKey()`. Уколико није притиснут ниједан тастер метод враћа вредност `null`. Стога, уколико је притиснут неки тастер (тј. `myKey!=null`), помоћу наредбе `switch` проверава се да ли је изабран неки од тастера горе, доле, лево, или десно и извршава одговарајућа наредба задата у поставци примера.

Потпуно решење задатка је представљено на слици 31.

```
public void act()
{
    String myKey;
    myKey = Greenfoot.getKey();

if(myKey!=null)
 {
     switch(myKey)
                {
                    case "left":
                    move(-7);
                    break;
                    case "right":
                    move(7);
                    break;
                    case "up":
                    turn(30);
                    break;
                    case "down":
                    turn(-30);
                    break;
                }
 }
}
```

Слика 31. Решење задатка из примера 11

## 8.3. Петља while

Петља **while** се користи за узастопно понављање једне или више наредби. Прецизније, петља **while** понавља наредбе, или блок наредби, све док је задати услов испуњен. У случају када услов више није испуњен, прекида се са извршавањем наредби које су у оквиру петље. Петља **while** има следећи облик у програмском језику *Java*:

```
while (logicki izraz){
naredbe_petlje;
}
```

Главни елементи петље **while** су:

- ✓ **logicki izraz** – тестира се на почетку сваког пролаза кроз петљу;
- ✓ **naredbe_petlje** – наредбе које се извршавају у случају да је логички израз тачан.

Петља **do-while** се примењује у случајевима када је потебно да се услов понављања тестира на крају петље уместо на почетку. Резервисана реч **do** се користи за означавање почетка петље. Петља **do-while** у програмском језику *Java* има следећи облик:

```
do {
naredbe_petlje;
} while (logicki izraz);
```

***Напомена:***

Знак „ **;** “ на крају **do-while** петље се не сме изоставити јер је то део наредбе. Изостављање би проузроковало синтаксну грешку.

### 8.4. Петља for

Петља `for` служи за решавање проблема који подразумевају узастопно понављање једне или више наредби. Петља `for` ради на следећи начин. Када петља започне рад, извршава се **иницијализација** приликом које се додељује почетна вредност управљачкој променљиви петље. Затим се испитује **услов** који пореди управљачку променљиву петље са задатом циљном вредношћу. У случају да је израз тачан, извршава се тело петље. Уколико је нетачан, петља се завршава. На крају сваког проласка кроз петљу долази до промене вредности управљачка променљиве петље на начин дефинисан одређеним изразом. Петља `for` у програмском језику *Java* има следећи облик:

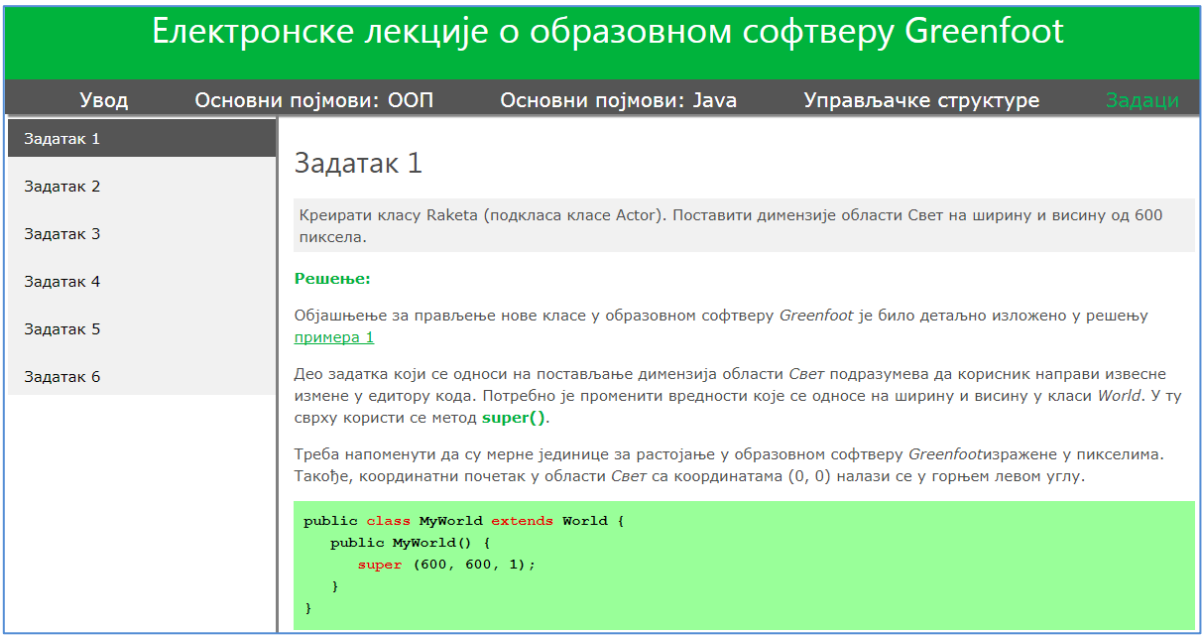
```
for (inicijalizacija; uslov; promena_vrednosti)
{
     naredbe_petlje;
}
```

Главни елементи петље `for` су:

- ✓ `inicijalizacija` – поставља почетну вредност управљачкој променљиви петље;
- ✓ `uslov` – логички израз који се тестира на почетку сваког проласка кроз петљу;
- ✓ `promena_vrednosti` – израз који се извршава се на крају сваког проласка кроз петљу и који доводи до променев редности управљачке променљиве;
- ✓ `naredbe_petlje` – наредбе које се извршавају при сваком проласку кроз петљу.

## 9. Задаци за самостални рад

У овом поглављу корисници ће имати прилику да провежбају неколико задатака који ће им помоћи да боље савладају принципе објектно оријентисаног програмирања, као и основе програмског језика *Java*. Сваки задатак ће имати решење. Задаци се такође налазе и у електронским лекцијама (Слика 32).




Слика 32. Електронске лекције - Задаци

***Задатак 1:*** Креирати класу *Raketa* (потклаcа класе *Actor*). Поставити димензије области *Свет* на ширину и висину од 600 пиксела.

**Решење:**

Објашњење за прављење нове класе у образовном софтверу *Greenfoot* је било детаљно изложено у решењу примера 1 у поглављу *4.1.*

Део задатка који се односи на постављање димензија области *Свет* подразумева да корисник направи извесне измене у едитору кода. Потребно је променити вредности које се односе на ширину и висину у класи *World*. У ту сврху користи се метод `super()`.

Треба напоменути да су мерне јединице за растојање у образовном софтверу *Greenfoot* изражене у пикселима. Такође, координатни почетак у области *Свет* са координатама (0, 0) налази се у горњем левом углу. Решење задатка приказано је на слици 33.

```
public class MyWorld extends World
{
      public MyWorld()
            {
                  super (600, 600, 1);
            }
}
```

Слика 33. Решење задатка 1

***Задатак 2 (допуна задатка 1):*** Објекат *Raketa* се креће по области *Svemir* тако да, уколико је притиснута стрелица на горе, објекат *Raketa* се помери унапред за 7 пиксела, а ако је притиснута стрелица на доле помери се уназад за 5 пиксела. Ако је притиснута стрелица лево, односно десно, објекат *Raketa* ће се окретати у леву, односну десну, страну за 7 степени.

**Решење:**

Решење овог задатка већ је обрађено у сличном примеру 7. Разлика између примера 7 и овог задатка је у томе што корисник мора да направи симулацију кретања по области коришћењем сва четири тастера са стрелицама на тастатури. Решење задатка је приказано на слици 34.

```
public class Raketa extends Actor
{
    public void act()
    {

       if (Greenfoot.isKeyDown("up")) move (7);
       if (Greenfoot.isKeyDown("left")) turn (-7);
       if (Greenfoot.isKeyDown("right")) turn (7);
       if (Greenfoot.isKeyDown("down")) move (-7);
     }
 }
```

Слика 34. Решење задатка 2

***Задатак 3 (допуна задатка 2):*** Уколико објекат *Raketa* наиђе на објекат *Robot* направити метод којим се *Robot* уклања из *Света* апликације.

**Решење:**

Задатак се може решити на начин изложен у решењу примера 8, али ће овде бити представљен алтернативни начин решавања. Да би се проверило да ли у околини ракете постоји робот може се користити метод `getOneObjectAtOffset()`. Овај метод враћа показивач на објекат типа *Robot* уколико се исти налази на растојању (0, 0) од ракете, у супротном враћа `null`. Ако је објекат *Raketa* налетео на робота (тј. `robot!=null`), робот ће бити уклоњен из света апликације коришћењем методе `removeObject()`. Овде треба истаћи да је у овом задатку употребљен оператор поређења (**!=**). Решење је приказано на слици 35.

```
Actor robot=getOneObjectAtOffset(0,0,Robot.class);
        if (robot!=null)
        {
            getWorld().removeObject(robot);
        }
```

Слика 35. Решење задатка 3

***Задатак 4 (допуна задатка 3):*** Поставити једну ракету на позицију са координатама (30, 30). Након тога, насумично поставити 10 робота у области Свет. Задатак решити применом петље `for`.

**Решење:**

Нови објекат *Robot* на позицију са координатама (30, 30) у области *Свет* може се додати применом оператора `new` на начин приказан на слици 35. Нови објекти типа *Robot* биће насумично додавани у област *Свет* применом петље `for`. Да би се то постигло потребно је декларисати две нове променљиве `x` и `y` које су целобројног типа. Ове две променљиве заправо представљају координате положаја сваког робота, које ће бити насумично изабране применом методе `getRandomNumber()`. Решење задатка дато је на слици 36.

```
public class MyWorld extends World
{
      public MyWorld()
      {
            super(600, 600, 1);
            addObject(new Robot(),30,30);

            for (int i=0; i<10; i++)
            {
                  int x,y;

                  x=Greenfoot.getRandomNumber(600);
                  y=Greenfoot.getRandomNumber(400);

                  addObject(new Robot(), x, y);
            }
      }
}
```

Слика 36. Решење задатка 4

***Задатак 5:*** Решити претходни задатак применом петље `while`.

**Решење:**

За решење овог задатка потребно је увести променљиву `br` у којој ће се чувати број додатих робота. Решење је дато на слици 37.

```
public class MyWorld extends World
{
      public MyWorld()
      {
            super(600, 600, 1);
            addObject(new Robot(),30,30);
            int br =0;
            while(br < 10)
            {
                  int x,y;

                  x=Greenfoot.getRandomNumber(600);
                  y=Greenfoot.getRandomNumber(400);

                  addObject(new Robot(), x, y);
                  br++;
            }
      }
}
```

Слика 37. Решење задатка 5

***Задатак 6 (допуна задатка 4):*** Ако је број ухваћених робота 5 зауставити апликацију. Објекти *Raketa* се крећу на начин описан у тексту задатка 2.

**Решење:**

За решење овог задатка потребно је декларисати нову променљиву `br` и доделити јој почетну вредност 0. Ова променљива представља бројач који ће се аутоматски увећавати за један (коришћењем унарног аритметичког оператора за инкрементацију) сваки пут када се објекат *Robot* уклони. Заустављање апликације врши се помоћу методе `stop()`. Решење задатка приказано је на слици 38.

```
public class Raketa extends Actor
{
    int br=0;
    public void act()
    {

       if (Greenfoot.isKeyDown("up")) move (7);
       if (Greenfoot.isKeyDown("left")) turn (-7);
       if (Greenfoot.isKeyDown("right")) turn (7);
       if (Greenfoot.isKeyDown("down")) move (-7);

       Actor robot=getOneObjectAtOffset(0,0,Robot.class);
       if (robot!=null)
       {
           getWorld().removeObject(robot);
         br++;
       }
        if (br==5) Greenfoot.stop();
     }
}
```

Слика 38. Решење задатка 6

***Задатак 7 (допуна задатка 6):*** Написати метод који омогућава да се објекти *Robot* померају 5 пиксела унапред, ако се кликне тастером миша. Сваки пут када дође до ивице области објекат *Robot* скреће под углом од 10 степени.

**Решење:**

У овом задатку неопходно је користити метод `mouseClicked()` који је имплементиран у библиотеци класе *Greenfoot*. Метод враћа вредност тачно ако и само ако је кликнуто мишем. Решење задатка је дато на слици 39.

```
public class Robot extends Actor
{
    public void act()
    {
       if (Greenfoot.mouseClicked(null)) move(5);
       if(isAtEdge()) turn(10);
     }
 }
```

Слика 39. Решење задатка 7

# 10. Закључак

Образовни софтвер *Greenfoot* на интерактиван начин омогућава корисницима стицање искуства у објектно оријентисаном програмирању, као и развијање алгоритамског начина размишљања. Оно што је јако битно, образовни софтвер *Greenfoot* не захтева унапред познавање програмирања те га могу користити и особе које се по први пут сусрећу са програмирањем. Наиме, образовни софтевер *Greenfoot* је осмишљен тако да кроз лак развој *2D* графичких апликација, као што су разне симулације и интерактивне игрице, корисницима приближи објектно оријентисано програмирање.

У овом раду на детаљан начин је представљено радно окружење образовног софтвера *Greenfoot.* Поред тога, кроз низ илустративних примера описано је како се образовни софтвер *Greenfoot* користи за креирање апликација. Имајући у виду да образовни софтвер *Greenfoot* омогућава креирање апликација у програмском језику *Java*, једном од најраспрострањенијих програмских језика данашњице, у раду су изложени и основни концепти програмског језика *Java.* Целокупно излагање је пропраћено одговарајућим примерима и задацима како би се основни концепти објектно оријентисаног програмирања и програмског језика *Java* додатно приближили читаоцу.

Као пратећи материјал овог мастер рада, развијене су електронске лекције. Због јавне доступности, електронске лекције дају могућност самосталног учења свима онима који су заинтересовани да науче основе објектно оријентисаног програмирања. Поред тога, електронске лекције могу бити коришћене од стране професора као материјал у току наставе, али и од стране ученика као приручник за самостални рад и учење након наставе. Требало би истаћи да се електронске лекције, развијене за потребе овог мастер рада, могу допунити напреднијим концептима објектно оријентисаног програмирања уколико то буде било потребно.

# Литература

[1] Kölling, M. (2010). The greenfoot programming environment. ACM Transactions on Computing Education (TOCE), 10(4), 14.

[2] Greenfoot : https://www.greenfoot.org/about/contributors.html

Pristupljeno novembra 2017.

[3] Greenfoot documentation: https://www.greenfoot.org/doc

Pristupljeno novembra 2017.

[4] Bates, B.,Sierra, K., (2009). Head first Java, 2nd Edition, O'Reilly Media, USA.