Creating Java Programs with Greenfoot

Using Randomization and Understanding Dot Notation and Constructors

ORACLE ACADEMY

Overview

This lesson covers the following topics:

- Create randomized behaviors
- Define comparison operators
- Create IF-ELSE control statements
- Create an instance of a class
- Recognize and describe dot notation

getRandomNumber Method

- The getRandomNumber method is a static method that returns a random number between zero and a parameter limit.
- This method is used to eliminate predictability in your program.
- Method signature:

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

Dot Notation

- New subclasses that you create do not inherit the getRandomNumber method.
- This method must be called from the Greenfoot class using dot notation.
- Example :

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

When you want to use a method but it is not inherited by the class you are programming, specify the class or object that has the method before the method name, then a dot, then the method name. This technique is called dot notation.

Dot Notation Format

- The format for dot notation code includes:
  - Name of class or object to which the method belongs
  - Dot
  - Name of method to call
  - Parameter list
  - Semicolon

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

Dot Notation Example

- The getRandomNumber method shown below:
  - Calls a random number between 0 and up to, but not including 15.
  - Returns a random number between 0 and 14.

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

Greenfoot API

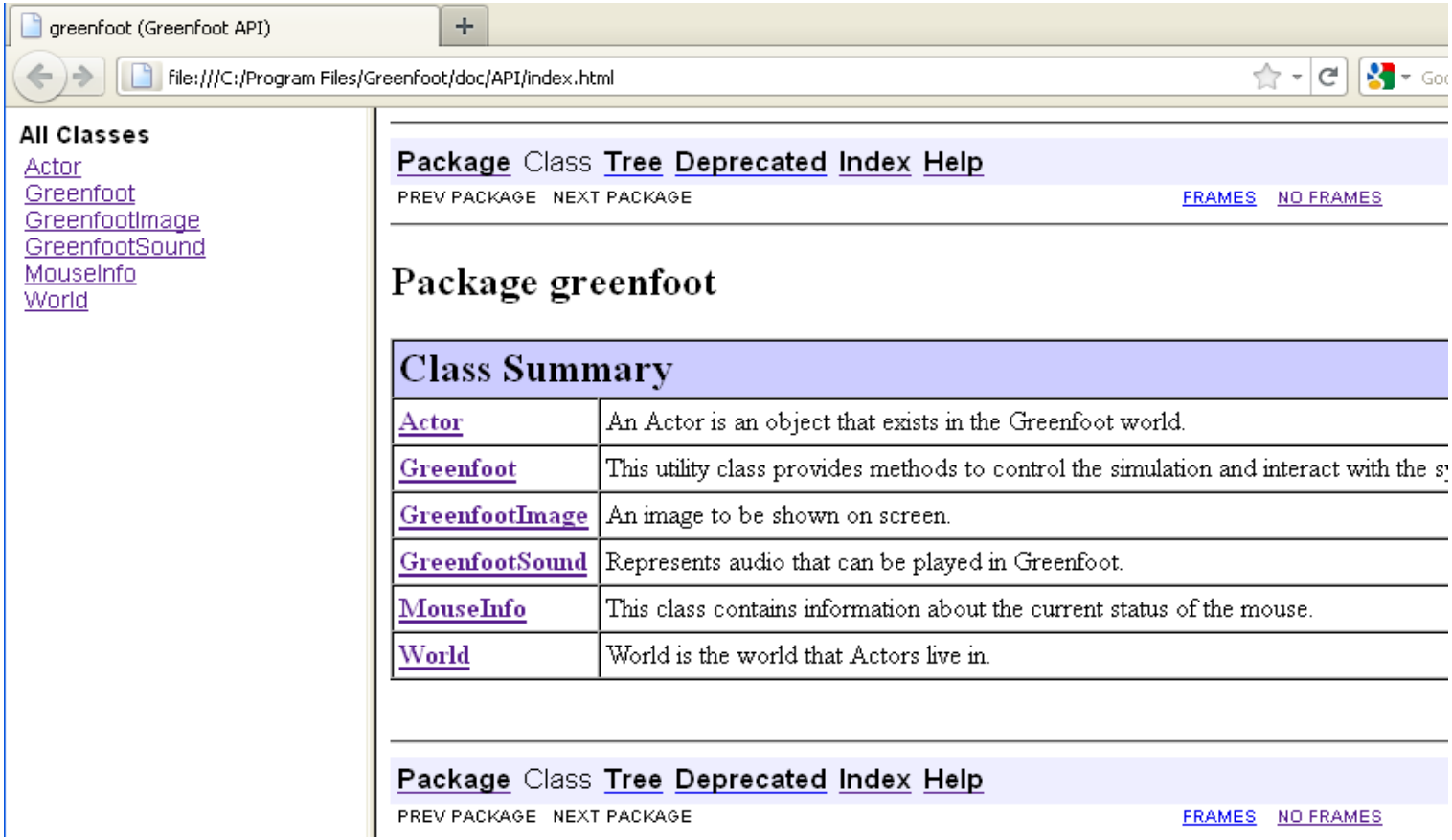
Reference the Greenfoot Application Programmers' Interface (API) to examine additional methods to call using dot notation.

The Greenfoot Application Programmers' Interface lists all of the classes and methods available in Greenfoot.

Steps to View Methods in Greenfoot Class

1. In the Greenfoot environment, select the Help menu.
2. Select Greenfoot Class Documentation.
3. Click the Greenfoot class.
4. Review the method signatures and descriptions.

Greenfoot API Interface



Comparison Operators

- Use comparison operators to compare a randomized value to another value in a control statement.
- The example below determines if the random number is less than 20. If it is, then the object turns ten degrees.

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

Comparison operators are symbols that compare two values.

Comparison Operator Symbols

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

Gaming Problem Solved with Random Behavior

- Problem: A banana object should move randomly so it is more challenging for the keyboard-controlled object, a monkey, to eat it.
- Solution:
  - The banana should turn a small amount as it moves.
  - To code this solution, turn the banana a random number of degrees, up to 20 degrees, 6% of the time as it moves.

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

Random Behavior Format

- The programming statement below includes:
  - IF control statement with the getRandomNumber method.
    - Parameter limit of 100.
    - Comparison operator <.
    - Number 6 to limit the range of values to return to 0-5.
  - Method body with statement to indicate that the object should turn up to 20 degrees if the condition is true.

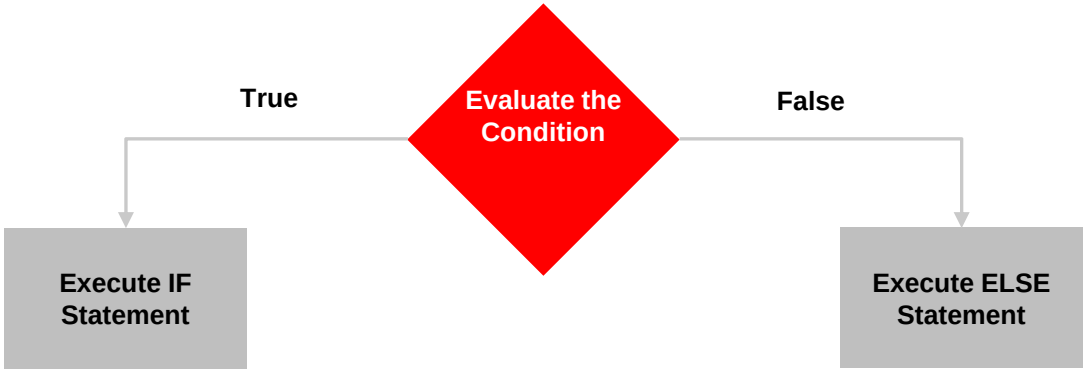
```
if (Greenfoot.getRandomNumber(100) < 6)
{
    turn(Greenfoot.getRandomNumber(20));
}
```

Conditional Behavior

- Instances can be programmed to perform specific behaviors if a condition is not met, using an IF-ELSE statement.
- For example, if an instance is programmed to turn 6% of the time, what does it do the other 94% of the time?

An IF-ELSE statement executes its first code segment if a condition is true, and its second code segment if a condition is false, but not both.

IF-ELSE Statement Execution



IF-ELSE Statement Format

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

IF-ELSE Statement Example

If a random number between 0-6 is selected, turn 10 degrees. Otherwise, turn 5 degrees.

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

Automate Creation of Instances

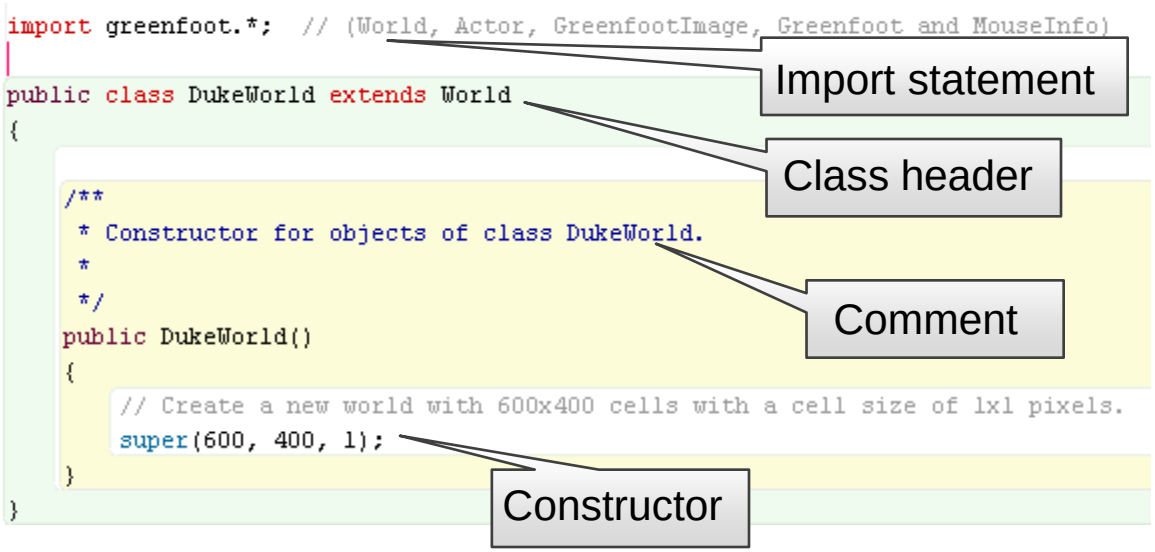
- Using the World subclass, actor instances can be programmed to automatically appear in the world when a scenario is initialized.
- In Greenfoot, the default behavior for instances is as follows:
  - The World subclass instance is automatically added to the environment after compilation or initialization of the scenario.
  - The Actor subclass instances must be manually added by the player.

Automate Creation of Instances in a Scenario

- Problem: When a Greenfoot scenario (such as leaves and wombats) is started, the instances must be manually added by the player to play the game.
- Solution: Program instances to be automatically added to the world when the scenario is initialized.

World Class Source Code

To understand how to automate creation of Actor instances, you need to understand how the World class source code is structured. The World constructor is used to automate creation of Actor instances when the scenario is initialized.



Constructors

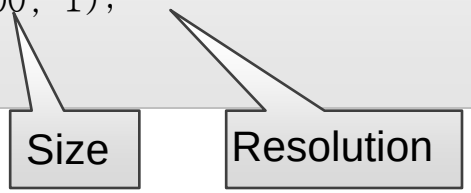
- Constructors:
  - Define the instance's size and resolution.
  - Have no return type.
  - Have the same name as the name of the class. For example, a World constructor is named World.

A constructor is a special kind of method that is automatically executed when a new instance of the class is created.

World Constructor Example

- The example constructor below constructs the World superclass instance as follows:
  - Size: x = 600, y = 400.
  - Resolution: 1 pixel per cell.
  - Keyword super in the constructor's body calls the superclass World for each instance of the DukeWorld subclass.

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



Automatically Create Actor Instances

This World constructor adds a Duke object at specified X and Y coordinates using the addObject method.

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

addObject Method

- The addObject method is a World class method that adds a new object to the world at specific x and y coordinates. It includes:
  - Keyword new to tell Greenfoot to create a new object of a specific class.
  - Method parameters:
    - Named object from Actor class.
    - Integer value of X coordinate.
    - Integer value of Y coordinate.
- Method signature:

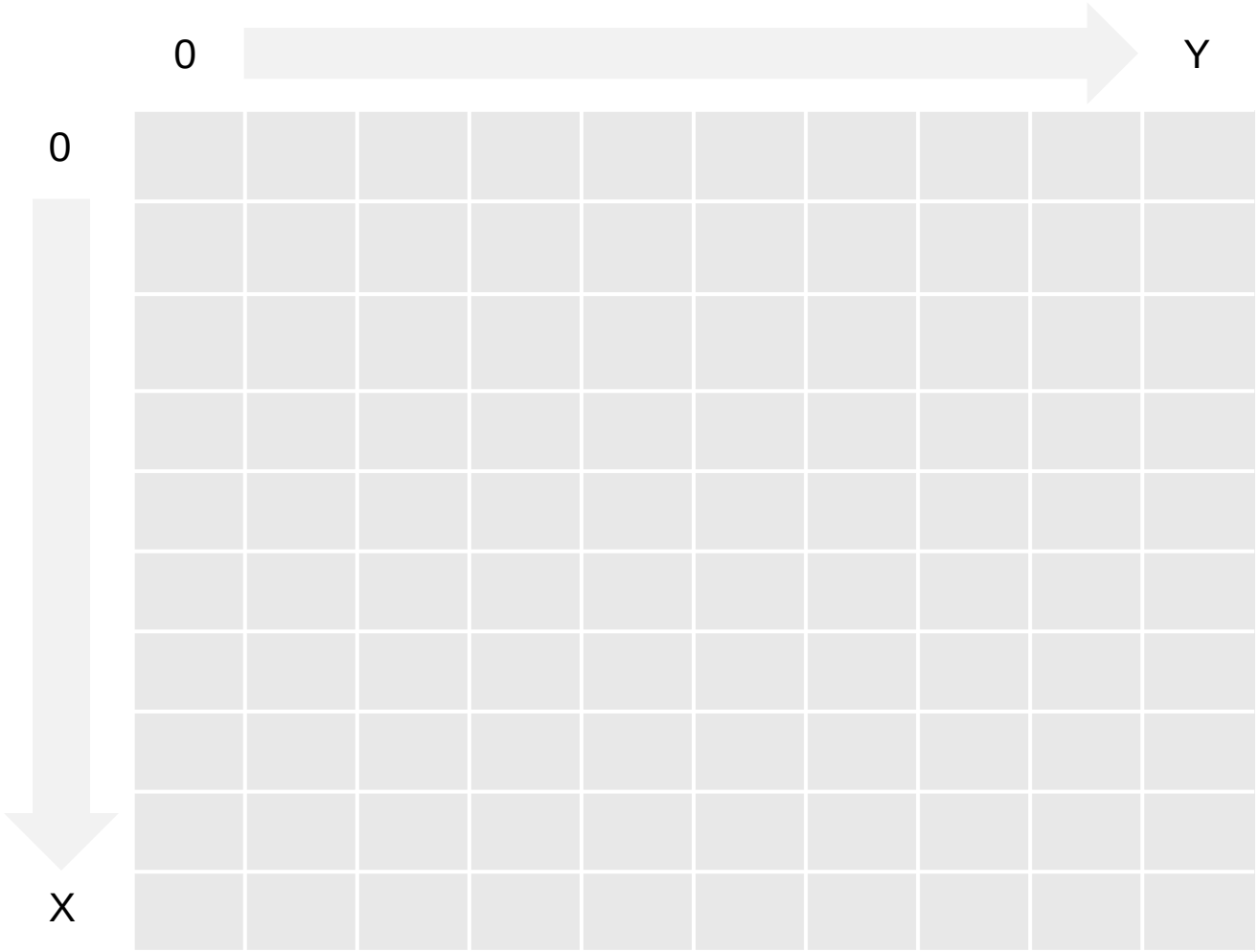
```
void addObject(Actor object, int x, int y)
```

new Keyword

- The new keyword creates new instances of existing classes.
- It starts with the keyword new, followed by the constructor to call.
  - The parameter list passes arguments (values) to the constructor that are needed to initialize the object's instance variables.
  - The default constructor has an empty parameter list and sets the object's instance variables to their default values.

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

Greenfoot World Coordinate System



Add Objects Using World Constructor Example

```
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);
    }
}
```

Terminology

Key terms used in this lesson included:

- Comparison operators
- Constructor
- Dot notation
- new Keyword

Summary

In this lesson, you should have learned how to:

- Create randomized behaviors
- Define comparison operators
- Create IF-ELSE control statements
- Create an instance of a class
- Recognize and describe dot notation

Practice

The exercises for this lesson cover the following topics:

- Creating randomized actor behavior using comparison operators
- Creating instances of a class