

The logo for Oracle Academy is centered on a light gray background. It features the word "ORACLE" in a bold, orange, sans-serif font. Below it, the word "Academy" is written in a smaller, dark gray, sans-serif font. The entire logo is framed by a thin black border, with dark gray horizontal bars at the top and bottom.

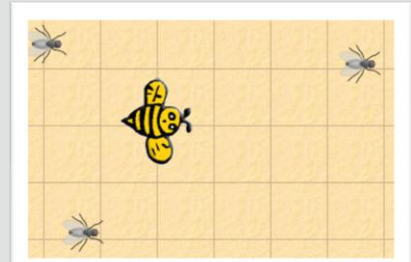
ORACLE

Academy

Java Fundamentals

3-5

Randomization and Constructors



ORACLE
Academy

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

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Objectives

- This lesson covers the following objectives:
 - 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)
```

Randomization allows us to create games that will vary each time we play them and consequently make them more enjoyable.

Static methods are methods that belong to a class rather than an instance.

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 :

```
public void act()  
{  
    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.



ORACLE
Academy

JF 3-5
Randomization and Constructors

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

5

To gain access to the `getRandomNumber` method we have to tell java it can be found in the Greenfoot class. So we write `Greenfoot.getRandomNumber()`.

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

```
className.methodName (parameters);  
objectName.methodName (parameters);
```



When referring to the class that we are coding in we can use the optional word "this" to represent the current class. So previously when we used `move(2)`, we could have said `this.move(2)`.

The "this" represents the object we are editing the code for, but is optional and is often left out.

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

```
public void act()  
{  
    Greenfoot.getRandomNumber(15);  
}
```

Greenfoot.getRandomNumber(15) means it will return one of 15 random numbers between and including 0-14.

What if we wanted a random number between 1 and 10? We can simply use
Greenfoot.getRandomNumber(10)+1;

Greenfoot API

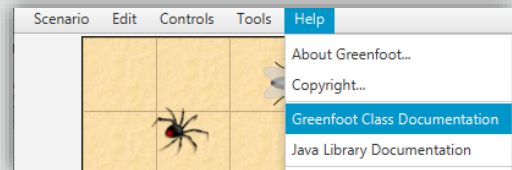
- Reference the Greenfoot Application Programmers' Interface (API) to examine additional methods to call using dot notation

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

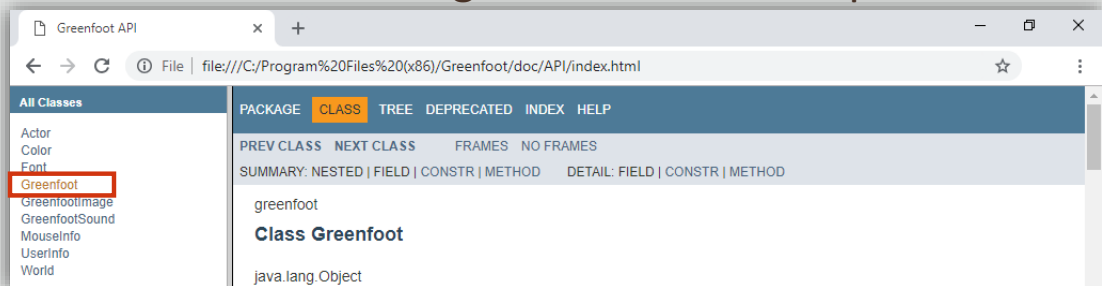


Steps to View Methods in Greenfoot Class

- In the Greenfoot environment, select the Help menu
- Select Greenfoot Class Documentation



- Click the Greenfoot class
- Review the method signatures and descriptions



Greenfoot API Interface

The screenshot shows a web browser window titled "Greenfoot API" with the URL `file:///C:/Program%20Files%20(x86)/Greenfoot/doc/API/index.html`. On the left, under "All Classes", a list includes Actor, Color, Font, Greenfoot, GreenfootImage, GreenfootSound, MouseInfo, UserInfo, and World. The main area, titled "Method Summary", displays the static methods of the Greenfoot class. It features three tabs: "All Methods" (selected), "Static Methods", and "Concrete Methods". Below these is a table with two columns: "Modifier and Type" and "Method and Description".

Modifier and Type	Method and Description
static java.lang.String	<code>ask(java.lang.String prompt)</code> Get input from the user (and freeze the scenario while we are waiting).
static void	<code>delay(int time)</code> Delay the current execution by a number of time steps.
static java.lang.String	<code>getKey()</code> Get the most recently pressed key, since the last time this method was called.
static int	<code>getMicLevel()</code> Get the microphone input level.
static MouseInfo	<code>getMouseInfo()</code> Return a mouse info object with information about the state of the mouse.
static int	<code>getRandomNumber(int limit)</code> Return a random number between 0 (inclusive) and limit (exclusive).
static boolean	<code>isKeyDown(java.lang.String keyName)</code> Check whether a given key is currently pressed down.

When creating actors we inherit from the Actor class.

When creating worlds we inherit from the World class. It is worth learning what methods are available in both.

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

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

Comparison operators are symbols that compare two values.



Remember the `getRandomNumber(100)` will generate a number between 0 and 99.

Comparison Operator Symbols

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

A common mistake when doing an Equals comparison is to only add one equals sign "=". This will then try to assign the second value to the first and in most cases generate a syntax error.

Gaming Problem Solved with Random Behavior

- Problem:
 - A Fly object should move randomly so it is more challenging for the keyboard-controlled object, a Bee, to catch it
- Solution:
 - The Fly should turn a small amount as it moves
 - To code this solution, turn the Fly a random number of degrees, up to 20 degrees, 10% of the time as it moves

```
public void act()
{
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        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 10 to limit the range of values to return to 0-9
- Method body with statement to indicate that the object should turn up to 20 degrees if the condition is true

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

Random Behavior Format

- The problem with the previous example is that the fly always turns in a clockwise circle
- We can change this by modifying the parameter in the turn method as shown below
- This will then generate a random number between -45 and +44
- Remember a negative turn value rotates counterclockwise

```
public void act()
{
    move(1);
    if (Greenfoot.getRandomNumber(100) < 10)
    {
        turn(Greenfoot.getRandomNumber(90)-45);
    }
}
```

You can use a few random numbers of your own to test the logic of the fly movement using paper and pencil if you are not convinced of the range that will come back.

Conditional Behavior

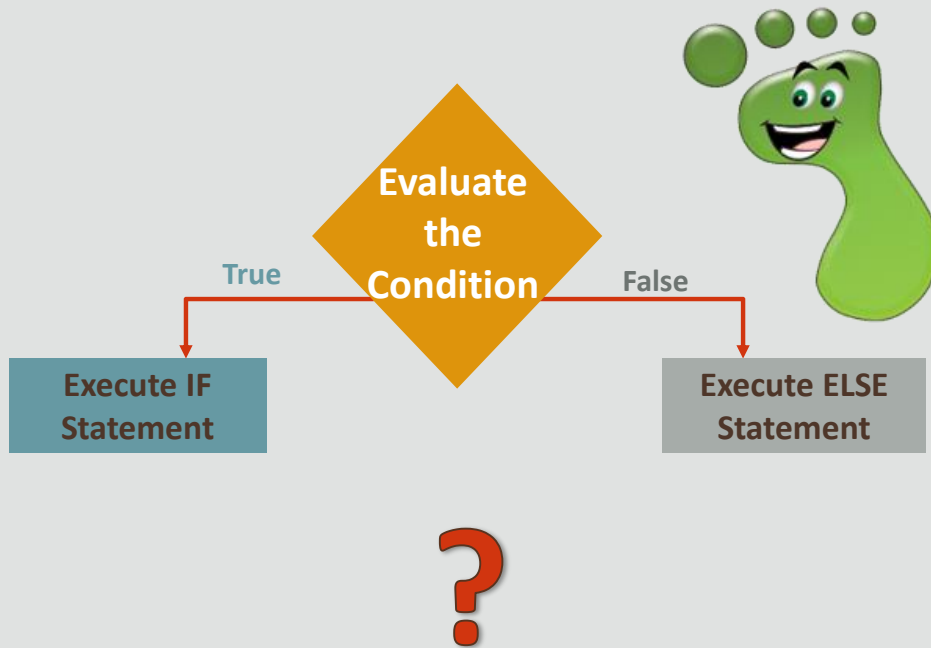
- Instances can be programmed to perform specific behaviors if a condition is not met, using an if-else statement
- For example, if an instance is programmed to turn 6% of the time, what does it do the other 94% of the time?

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.



We have to decide whether we require two IF statements or one if-else statement. If you want both of the code sections to possibly run then we have two IF statements. If you only ever want one or the other to run, then you would use an if-else statement.

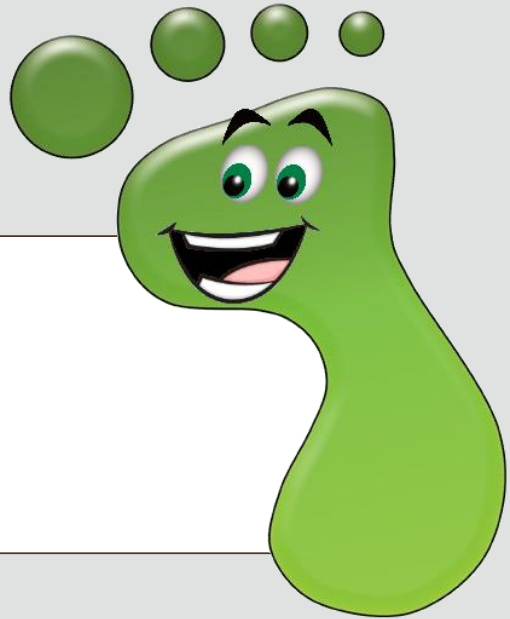
if-else Statement Execution



In an IF – ELSE statement only one of the code statements will run.

if-else Statement Format

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



Notice we have no semi-colon after the brackets of an IF command.

if-else Statement Example

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

```
public void act()
{
    move(1);
    if (Greenfoot.getRandomNumber(100) < 7)
    {
        turn(10);
    }
    else
    {
        turn(5);
    }
}
```

Automate Creation of Instances

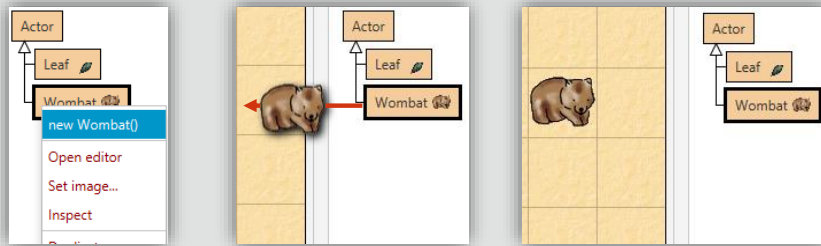
- Using the World subclass, actor instances can be programmed to automatically appear in the world when a scenario is initialized
- In Greenfoot, the default behavior for instances is as follows:
 - The World subclass instance is automatically added to the environment after compilation or initialization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



We will see later we have a quicker method using the "Save the World" command.

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

- The default name for the World subclass is MyWorld

```
public class MyWorld extends World
{
    /**
     * Constructor for objects of class MyWorld.
     *
     */
    public MyWorld()
    {
```

- To clearly identify your current World it can be useful to rename it
- Update all occurrences of MyWorld in the source code to BeeWorld

```
public class BeeWorld extends World
{
    /**
     * Constructor for objects of class BeeWorld.
     *
     */
    public BeeWorld()
    {
```

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

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

Class

header

Comment

Constructor

A constructor is normally defined by

public <classname>()

In the example above we have public BeeWorld()

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.

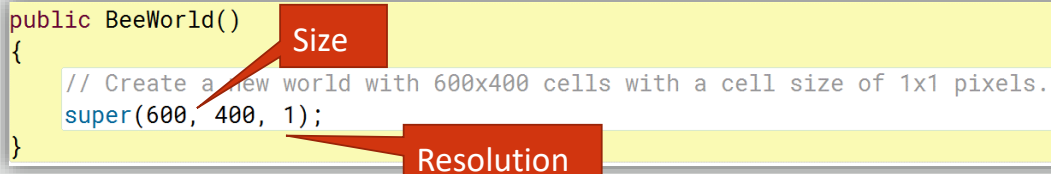


Constructors will be explored in greater depth later, but they provide an excellent mechanism for setting up default values for your class fields in the object instance.

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 constructor for each instance of the BeeWorld subclass

```
public BeeWorld()  
{  
    // Create a new world with 600x400 cells with a cell size of 1x1 pixels.  
    super(600, 400, 1);  
}
```

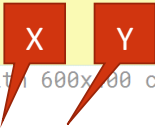


We can quite easily modify the values of the World here. Once changed your scenario will reflect the new size after a compilation.

Automatically Create Actor Instances

- This World constructor adds a new Bee object at specified X and Y coordinates using the addObject() method

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

A diagram consisting of two red squares, one labeled 'X' and one labeled 'Y', with red arrows pointing from them to the numerical values '150' and '100' respectively in the 'addObject' line of the code block.

As the constructor BeeWorld() is only called when the BeeWorld is created this code only runs once.

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 definition:

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

When new <classname>() is called it looks for a constructor for that class. If none exists then it defaults the class field values to their type default. i.e. integers become 0

new Keyword

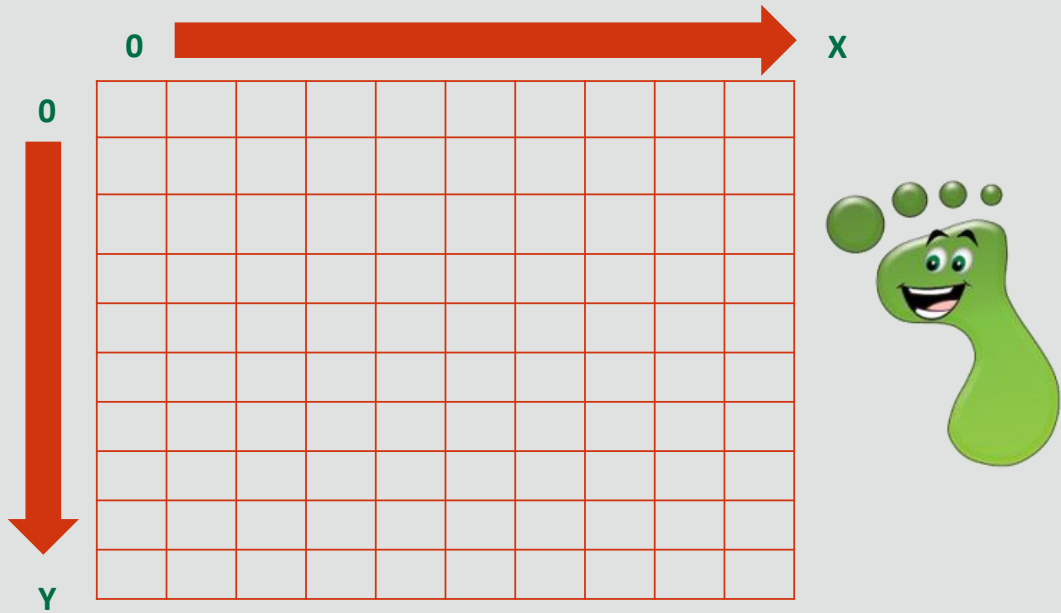
- The new keyword creates new instances of existing classes
- It starts with the keyword new, followed by the constructor to call
 - The parameter list passes arguments (values) to the constructor that are needed to initialize the object's instance variables
 - The default constructor has an empty parameter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



```
new ConstructorName()
```

If you define no constructor for your classes then a default one is generated by the Java compiler. You will not see this in the code.

Greenfoot World Coordinate System



ORACLE
Academy

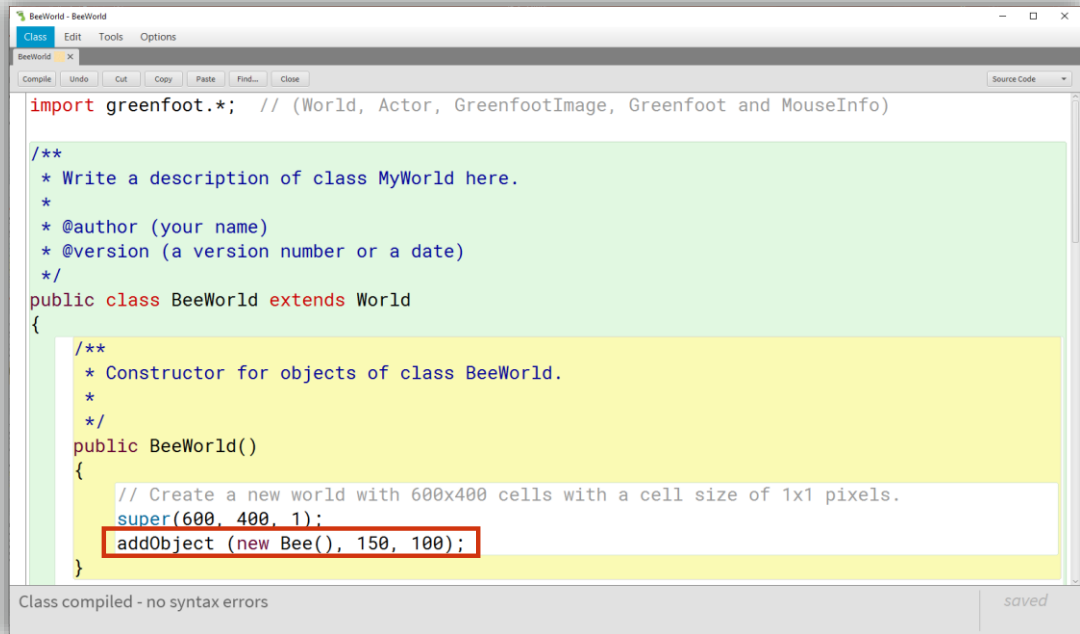
JF 3-5
Randomization and Constructors

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

29

Top left is the point (0,0) and bottom right is the dimensions of your world.

Add Objects Using World Constructor Example



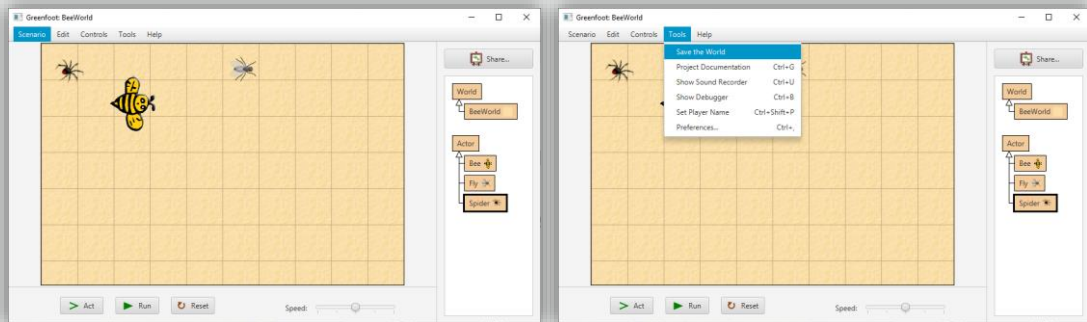
```
import greenfoot.*; // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class MyWorld here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BeeWorld extends World
{
    /**
     * Constructor for objects of class BeeWorld.
     *
     */
    public BeeWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);
        addObject(new Bee(), 150, 100);
    }
}
```

We can see a new instance of the Bee class being created and position at the point (150, 100).

Save the World

- Greenfoot has an additional way to setup the initial placement of Actors using "Save the World"
- We can place our actors around the world manually in an initial position
- Then select Tools -> Save the World



ORACLE
Academy

JF 3-5
Randomization and Constructors

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

31

Save the World

- This creates a new method in your world called `prepare()` and creates a call to this within the constructor
- The `prepare` method will create instances of the actor then add them to the world at the location you placed them
- This is useful if you have lots of objects to place

```
public BeeWorld()  
{  
    // Create a new world with 600x400  
    super(600, 400, 1);  
    addObject (new Bee(), 150, 100);  
    prepare();  
}
```

```
/**  
 * Prepare the world for the start of the program.  
 * That is: create the initial objects and add them to the world.  
 */  
private void prepare()  
{  
    Fly fly = new Fly();  
    addObject(fly, 334, 42);  
    Spider spider = new Spider();  
    addObject(spider, 46, 48);  
}
```

ORACLE
Academy

JF 3-5
Randomization and Constructors

Copyright © 2022, Oracle and/or its affiliates. Oracle, Java, and MySQL are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

32

`Bee bee = new Bee()` creates a new `Bee` instance that can be accessed via the reference variable called `bee`. Remember that Java is case sensitive, so `Bee` and `bee` are treated as different. So rather than say `addObject (new Bee(), 100,100)`, this is replaced with `Bee bee = new Bee();`
`addObject(bee, 100,100);`
This does give us some additional options that we will explore later.

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
 - Save the World feature



