ORACLE

Academy

Java Fundamentals

3-8

World, Animation, and Game End

```
private void caughtBySpider() {
    if (isTouching(Spider.class)) {
        setLocation(20,20);
        lives--;
        if (lives < 0) {
            Greenfoot.stop();
        }
    }
}
```

ORACLE
Academy

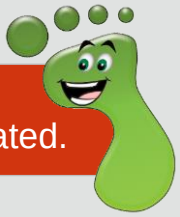
Objectives

- This lesson covers the following objectives:
  - Construct a world object using a constructor method
  - Create an object using a constructor
  - Write programming statements to use the new keyword
  - Define the purpose and syntax of a variable
  - Recognize the syntax to define and test variables
  - Write programming statements to switch between two images
  - Write programming statements to end a game

Constructors

- When a new World subclass is created and compiled, Greenfoot executes a constructor that creates an instance of it to display in the scenario
- Constructors set up the instance and establish an initial state, such as the size and resolution of the instance
  - Constructors have no return type
  - Their name, immediately following the word "public," is the same as the class in which they are defined

Constructors are special methods that are executed automatically whenever a new instance of the class is created.



As we saw previously we declare a constructor as
public <ClassName>(optional parameters)

Constructor Parameters

- A constructor's parameters allow an instance's initial values to be passed into the constructor
- These parameters:
  - Are only available to the instance created by the constructor
  - Have a restricted scope limited to when the constructor is declared
  - Have a restricted lifetime limited to the single execution of the constructor
  - Disappear once a constructor is finished executing
  - Are valid variables as long as the instance exists

In Greenfoot the constructor in your world subclass is added by default. Your actor subclasses do not have a constructor added in the code.

Constructor Example

- This constructor in the World subclass uses the super() keyword to pass the world's height, width and resolution values to the instance

```
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);

    }
```


"super" in Java is a call to the parent's constructor. So in BeeWorld we have super being added with 3 parameters setting the width, height and cell size.
This tells us that the World class has a constructor that accepts three integer values.

Parameters Example

- To change the size of the game board, modify the arguments in the parameter of the constructor
- This example makes the world square instead of rectangular by changing the x coordinate limit to 400

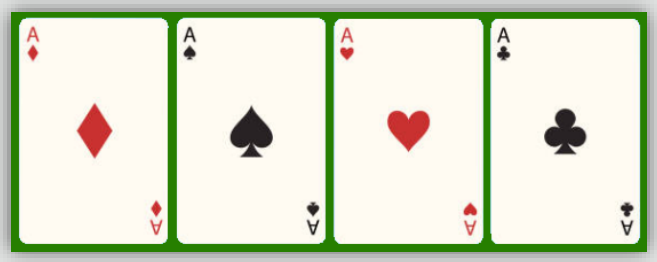
```
public class BeeWorld extends World
{
    /**
     * Constructor for objects of class BeeWorld.
     *
     */
    public BeeWorld()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(600, 400, 1);

    }
```


Always remember when designing your game that other users may not have a screen size that matches your resolution. So if you create a game of 1900 x 1200 dimensions, then lots of users will not be able to view all of this.

Automatically Create Actor Instances

- Write code in the World constructor to automatically add Actor instances to the game when the scenario is initialized
- This eliminates the need for the player to have to manually add instances before the game starts
- For example, in a matching game, the cards should automatically display in the scenario when the game starts



ORACLE Academy

We have seen previously that the "Save The World" option does exactly this, but we can modify this in the source code to extend its feature.

Code to Automatically Create Instances

- The code in the World constructor includes the following components:
  - super() statement with the size of the world as arguments
  - addObject() method with the following arguments:
    - Keyword new, followed by the class name, tells the constructor that a new instance of that class should be added
    - X and Y coordinates where the new instance should be positioned in the world

```
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);
}
```


Super(560,560,1) – creates a world of width 560 and height 560.

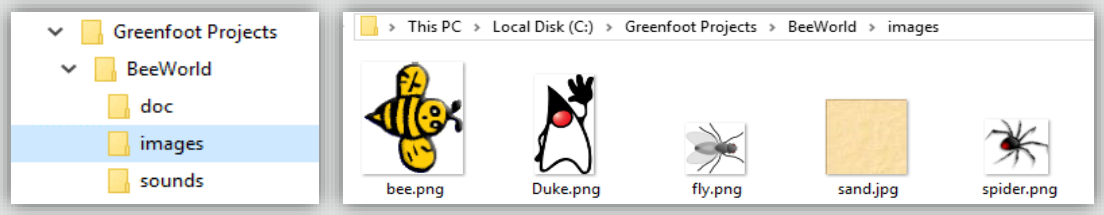
Greenfoot Actor Instances

- Alternating between two images that look slightly different gives an instance the appearance of movement
- Greenfoot Actor instances:
  - Receive and hold an image from their class
    - The image was assigned to the class when the class was created
  - Have the ability to hold multiple images
  - Can be programmed to change the image they display at any time

We do not have to limit ourselves to only 2 images. The more images we use in animation the smoother the effect.

GreenfootImage Class

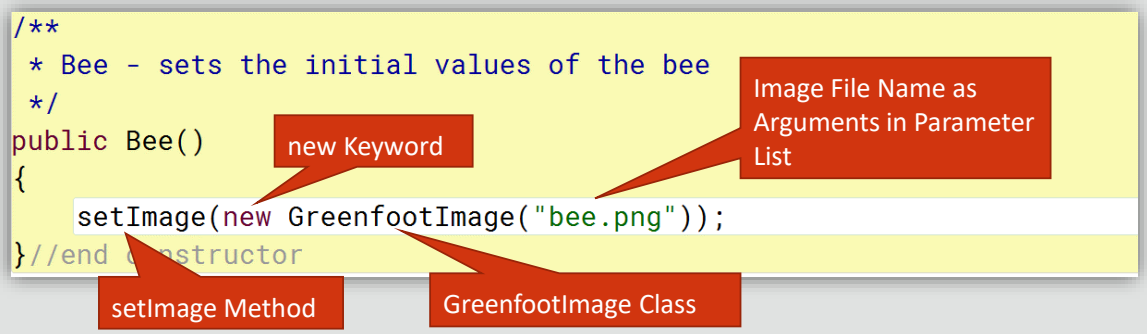
- The GreenfootImage class enables Greenfoot actors to maintain their visible image by holding an object of type GreenfootImage
- This class is used to help a class obtain and manipulate different types of images
- Images that this class will use must pre-exist in the scenario's Images folder



Remember that transparent images allow us to create more realistic effects, rather than the image looking like a block.

Constructor to Obtain New Image Object

- Create a constructor that retrieves a new image object from a file when creating an instance of a class
- The example constructor below creates the new image and attaches it to the Actor class



The new keyword will generate a call to the GreenfootImage constructor. In this example the constructor will accept a string value that relates to the name of an image stored in the Images folder

Assigning a New Image to a Class

- The statement below creates the new image object from the named image file
- When inserted into the class's source code, this image object is ready for the class to use
- The statement is executed as follows:
  - The GreenfootImage object is created first
  - The setImage() method call is executed, passing the newly-created image object as an argument to the parameter list

```
public Bee()
{
    setImage(new GreenfootImage("bee.png"));
}//end constructor
```


Remember that Java is case sensitive so the case of the letters in the string "bee.png" is important.

Assigning an Image Example

- The setImage() method assigns the image in the file "bee.png" to the Actor class
- Each time an instance of this class is added to the scenario, it displays the "bee.png" image

```
/**
 * Bee - sets the initial values of the bee
 */
public Bee()
{
    setImage(new GreenfootImage("bee.png"));
}//end constructor
```

Creates the new image

Image file name as argument

Allows image object to be used by Actor class
Expects image as parameter

Image from Greenfoot class

Why Instances Hold Multiple Images

- You may want an instance to hold and access multiple images:
  - To appear to change color
  - To appear to change from one type of object to another For example, magically change from a rabbit to a tortoise
  - To appear to move:
    - To walk:
      - Change from an object with left leg extended, to one with right leg extended
    - To flip cards:
      - Change from a blank card to a non-blank card
    - To fly:
      - Change from outstretched wings to folded wings

The hardest part of animation in Greenfoot or any other software tool is either drawing the images or gaining access to relevant images on the internet.

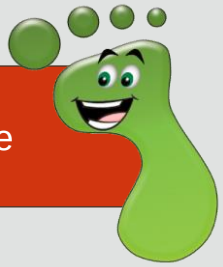
Accessing Multiple Images

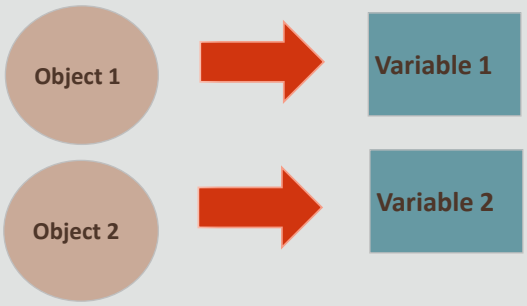
- For example, an instance could access two images, each with a slightly different wing position, so the instance flaps its wings as it moves
- To achieve this motion:
  - Create two images of the instance, each with slightly different wing positions
  - Store the two images in the instance, so they can be accessed repeatedly as the object moves
  - Code the class to alternate between the two images that are displayed

Variables

- Use class variables to store the two image objects
- This allows the class to easily access them for use within the instances

A variable is declared in a class. It is used to store information for later use, or to pass information. It can store objects or values.





ORACLE Academy

We have only created variables of type int or boolean. To store an image we use the type GreenfootImage.

Variable Format

- A variable's format includes:
  - Data type:
    - What type of data to store in the variable
  - Variable name:
    - A description of what the variable is used for so it can be referred to later

```
private variable-type  variable-name;
```

- In this example, the variable name is image1 and the variable type is GreenfootImage

```
private GreenfootImage image1;
```

ORACLE Academy

We normally declare class variables as private so that they cannot be accessed directly outside the class. We would create methods that would allow getting and setting of its value.

Declaring Variables

- Declare variables before the constructors and methods
- The format for declaring a variable includes:
  - Keyword private to indicate that the variable is only available within the Actor class
  - Class to which the image belongs
  - Placeholder for the variable into which the image will be stored

```
public class Bee extends Actor

{
    private GreenfootImage image1;
    private GreenfootImage image2;

    /**
     * Bee - sets the initial values of the bee
```

Variables

We can see we have added 2 additional class fields, both of type Greenfootimage. If we were to right click on a Bee instance and select inspect we would see these 2 fields.

Assignment Statements

- An assignment is needed to store objects in a variable
- When an object is assigned to a variable, the variable contains a reference to that object
- An assignment statement:
  - Stores the object or value into a variable
  - Is written with an equals symbol
- Format:

```
variable = expression;
```


A single equals = is an assignment and a double equals == is a comparison.

Assignment Statement Components

- An assignment statement includes:
  - Variable: Name of variable to store object or value
  - Equals symbol, which is the assignment symbol
  - Expression:
    - Name of object or value to assign
    - An instruction that the object or value is new
    - The class to which the image belongs
- Example:

```
public Bee()
{
    image1 = new GreenfootImage("bee.png");
    image2 = new GreenfootImage("bee2.png");
 }//end constructor
```


All of our images should be in the Images folder.

Initializing Images or Values

- Initializing is the process of establishing the instance and its initial values
- When the class creates new instances, each instance contains a reference to the images or values contained in the variables
- Guidelines:
  - Signature does not include a return type
  - Name of constructor is the same as the name of the class
  - Constructor is automatically executed to pass the image or value on to the instance when an instance of the class is created

Actor Constructors Example

- The following actor constructor tells Greenfoot to automatically create a new Bee instance and initialize, or assign, two variables to the instance
- The last line of the constructor, setImage(image1), indicates that the first variable should display when the instance is added to the scenario

```
private GreenfootImage image1;
private GreenfootImage image2;

/**
 * Bee - sets the initial values of the bee
 */
public Bee()
{
    image1 = new GreenfootImage("bee.png");
    image2 = new GreenfootImage("bee2.png");
    setImage(image1);
 }//end constructor
```

ORACLE Academy

We have used 2 images here, but we could use as many images as we want.

Test Values of Variables

- Once the class has initialized the two variables with the images, program the instance to automatically switch the image it displays as it moves
- As these images alternate with each movement, it makes the instance appear more animated
- It is possible to program the switch between images without having to write many lines of code that associates each image to every single movement

Write Actions in Pseudocode

- Identify the actions to program by writing them in pseudocode
- Pseudocode expresses the tasks or operations for the instances to perform in a mix of Java language and plain English words
- This helps to better understand what behaviors the instances should perform before writing the real code

Pseudo code allows you to focus on the algorithm rather than the syntax.

Pseudocode Example

- image1 is displayed when the instance is created
- When Bee makes its next movement, image2 should be displayed, and vice versa
- This is expressed as an if-else statement

```
if (current image displayed is image1) then
    use image2 now
else
    use image1 now
```

ORACLE Academy

Having a clear idea of your pseudocode makes writing the equivalent Java statements a lot easier. It is easy to become too focused on the syntax rather than thinking about the algorithm.

'==' Operator

- The programming statements that instruct the instance to alternate between images contains:
  - if-else statement
  - '==' operator (two equals signs)
- The '==' operator:
  - Is used in an if statement to test whether two values are equal
  - Compares one value with another
  - Returns a boolean (true or false) result
- Remember that '=' is the assignment symbol, not the symbol to test whether two values are equal

The java compiler will nearly always report an error if you mix up equals and double equals.

Components of if-else Statement

- Components of the if-else statement:
  - Method getImage receives the instance's current image
  - The '==' operator checks that the value the instance displayed is equal to image1
    - if equal, then display image2
    - else, display image1

getImage is a method of the Actor class.

if-else Statement Example

- The if-else statement below is written in the act method to make the instance alternate the display of two images as it moves forward

```
/**
 * 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(getImage()==image1)
        setImage(image2);
    else
        setImage(image1);
    //endif
    handleMovement();
    catchFly();
    turnAtEdge();
}//end method act
```


Notice that we have methods that we have created earlier listed at the bottom of the act method. We moved them to their own methods to make the code more readable.

if-else Statement Example

- We will also move the animation code away to its own method to keep the code cleaner

```
public void act()
{
    animateBee();
    handleMovement();
    catchFly();
    turnAtEdge();
}//end method act

private void animateBee(){
    if(getImage()==image1)
       setImage(image2);
    else
       setImage(image1);
    //endif
}//end method animateBee
```


Running the program probably produces an animation that is too fast. We could add a delay by using a counter that will only change the animation when the counter resets.

End the Game

- The Greenfoot class has a stop() method that you can use to end your game at a point that you designate
- You may want to end the game when:
  - The player achieves a milestone
  - Time runs out on the clock
  - The instance touches a certain coordinate or object

Alternatively you may not wish to stop the Greenfoot game, but return to a start screen instead.

Example Bee Game

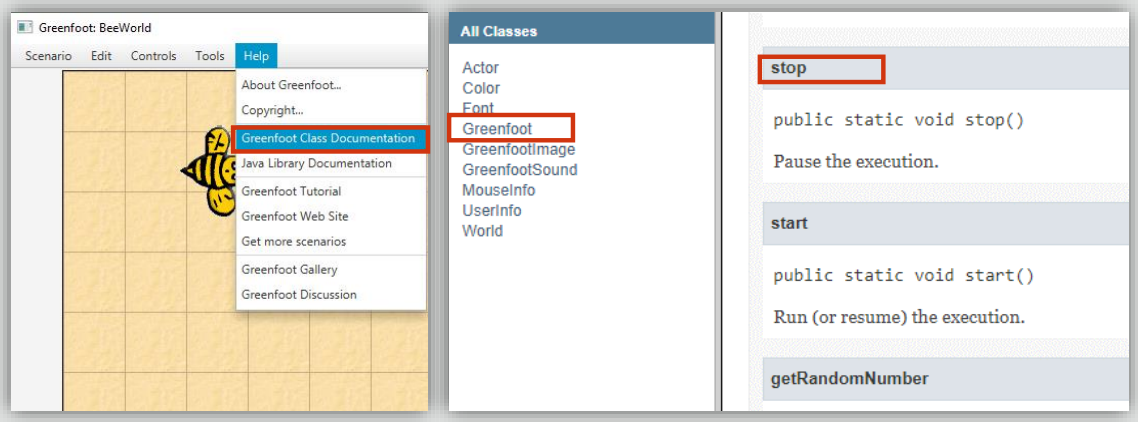
- Example game:
  - The player decides how many times the Bee is caught by the Spider object to end the game
  - When the game ends, a sound plays, "Game Over"
- Game specifications:
  - Create and initialize variables to store lives and score
  - Provide a count of the total Flies eaten (score)
  - Enter the stop() method to stop the game when the player's lives reach 0

ORACLE Academy

You can record your own Game Over .wav sound file.

Find stop() Method in Greenfoot API

- Go to the Help menu and select Greenfoot Class Documentation
- Find the stop() method in the Greenfoot API



ORACLE Academy

We can call stop() from either an Actor class or a World class.

Write stop() Method in Source Code

- At the point that the game should end, write the method as follows into the source code
- Dot notation is used to call the method

```
private void endGame(){
    Greenfoot.stop();
}//end method endGame
```

Assign Variables to Instances Example

- Bee must catch a number of Fly objects to increase the score
- The Bee will also lose a life if caught by the spider
- The variables are defined before the constructors and methods
- The Bee constructor assigns the variables to the instances it produces

```
public class Bee extends Actor

{
    private GreenfootImage image1;
    private GreenfootImage image2;
    private int score;
    private int lives;

    /**
     * Bee - sets the initial values of the bee
     */
    public Bee()
    {
        image1 = new GreenfootImage("bee.png");
        image2 = new GreenfootImage("bee2.png");
        setImage(image1);
        score = 0;
        lives = 3;
    }//end constructor
```


It is also good practice to add comments to your class variables so that the meaning is better understood by the reader. In this example they have been omitted due to space for the screenshot.

catchFly() Defined Method Example

- The catchFly() defined method is written below the act() method to tell the Bee to catch fly objects
- We will add one to the score variable for every Fly that is eaten

```
/**
 * catchFly - if the Bee touches a fly the fly is removed
 * A sound is played and a new fly is added to the game
 */
private void catchFly(){
       if(isTouching(Fly.class)){
           removeTouching(Fly.class);
           Greenfoot.playSound("slurp.wav");
           score++;
           getWorld().addObject(new Fly(), Greenfoot.getRandomNumber(getWorld().getWidth()),
                                           Greenfoot.getRandomNumber(getWorld().getHeight()));
        }//endif
}
```


The code score++; is the same as writing score = score+1; which increments the score by 1. The last line of the code gets the current world and calls its addObject() method that we saw earlier. This time we call it from an actor, so the getWorld() returns a reference to the current world. This means that every time we catch a Fly, another re-appears.

Assign Variables to Instances Example 2

- If the Bee comes into contact with the spider, then it should lose a life
- We will also re-position the Bee to the top left
- We will extend the Bee class by adding a new method – caughtBySpider() and adding a call to this in the act() method
- We will then test if the user has no more lives and stop the game

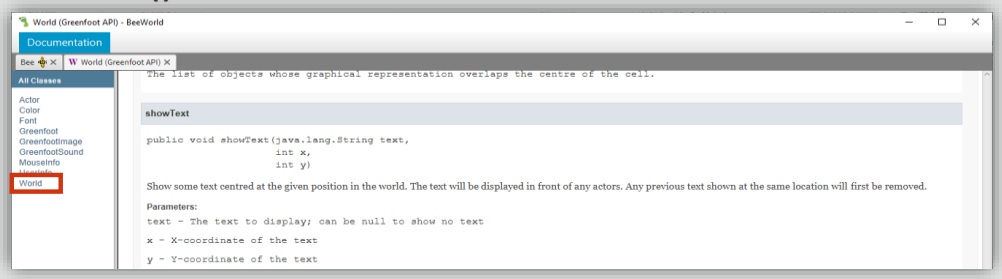
```
private void caughtBySpider(){
    if(isTouching(Spider.class)){
        setLocation(20,20);
        lives--;
        if(lives<0){
            endGame();
        }//endif
    }//endif
}//end method caughtBySpider

private void endGame(){
    Greenfoot.stop();
}//end method endGame
```


We could have written :
this.setLocation(20,20);
this.lives--;
Remember that "this" is optional and in most cases is not required.

Showing Text

- Sometimes we want to keep the user of an application informed on particular aspects of their interaction such as lives, score or cards left
- Greenfoot again has a number of ways to achieve this
- The simplest is by using the World method – showText()



The Greenfoot.showText() method was added in version 2.4.0

Update catchFly() Method

- We are going to add code to the method catchFly() to increment the score field and then display the result to the screen
- We have also moved the update score to its own method and called it within catchFly()

```
private void catchFly(){
        if(isTouching(Fly.class)){
            removeTouching(Fly.class);
            Greenfoot.playSound("slurp.wav");
            updateScore();
            getWorld().addObject(new Fly(), Greenfoot.getRandomNumber(getWorld().getWidth()),
                                            Greenfoot.getRandomNumber(getWorld().getHeight()));
         }//endif
}

private void updateScore(){
    score++;
    getWorld().showText("Score : " + score, 60, 390);
}//end method updateScore
```


Creating a new method called updateScore() allows us to call this from anywhere else in the Bee class. That means if we added something else that increased the score we could simply call updateScore() without having to re-write the code. We could also add a parameter like int scoreincrease that would increment the score by more than one –

```
updateScore(int scoreincrease) {
    score = score + scoreincrease;
}
```

Terminology

- Key terms used in this lesson included:
  - Constructor
  - Defined variable
  - Pseudocode

Summary

- In this lesson, you should have learned how to:
  - Construct a world object using a constructor method
  - Create an object using a constructor
  - Write programming statements to use the new keyword
  - Define the purpose and syntax of a variable
  - Recognize the syntax to define and test variables
  - Write programming statements to switch between two images
  - Write programming statements to end a game


ORACLE
Academy