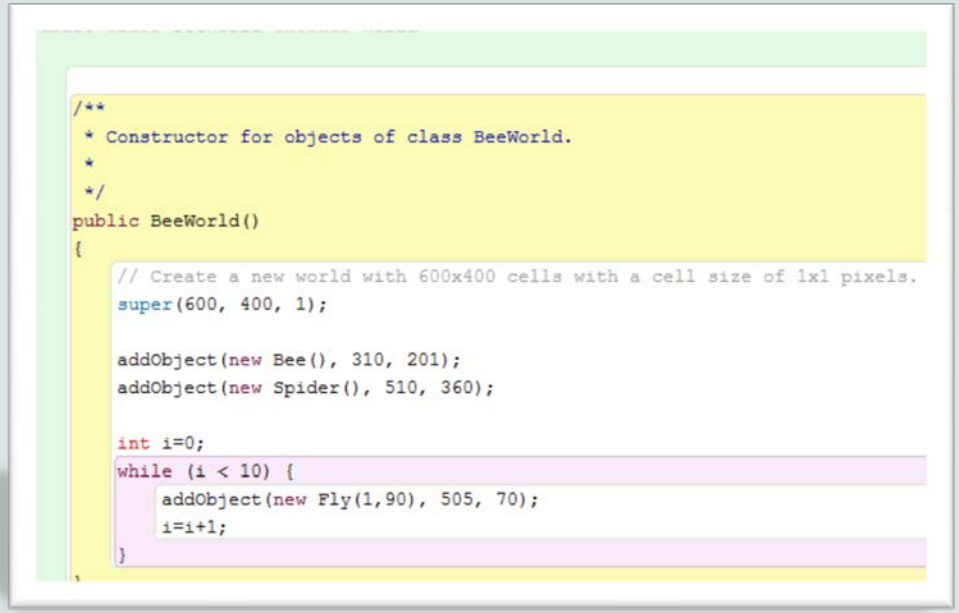

ORACLE
ACADEMY

Java Fundamentals

3-10
Loops, Variables, and Arrays



```
/**
 * 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(), 310, 201);
    addObject(new Spider(), 510, 360);

    int i=0;
    while (i < 10) {
        addObject(new Fly(1,90), 505, 70);
        i=i+1;
    }
}
```

ORACLE ACADEMY

Objectives

This lesson covers the following objectives:

- Create a while loop in a constructor to build a world
- Describe an infinite loop and how to prevent one from occurring
- Use an array to store multiple variables used to create a world
- Create an expression using logic operators
- Describe the scope of a local variable in a method

Using Loops

- Writing programming statements in the World constructor is an efficient way to create new instances with parameters passed to them.
- However, a more efficient way to create multiple instances is to use a loop.

A loop is a statement that can execute a section of code multiple times. There are several types of loops in Java programming.

while Loop

- The while loop executes a statement or set of statements a number of times while a condition is true.
- For example, with a while loop, you can:
  - Create 50 instances at once.
  - Execute a method 10,000 times.
  - Create an instance every time until the "s" key is pressed.

while Loop Components

- Components of a while loop:
  - Java keyword while
  - Condition in parentheses
  - One or more statements

```
while (condition)
{
    statement;
    statement;
}
```

Control Execution of while Loop

- Components to control how many times the while loop is executed:
  - Loop variable: A counter that tells how many times to execute the loop (often named i)
  - Control operators
  - Local variable

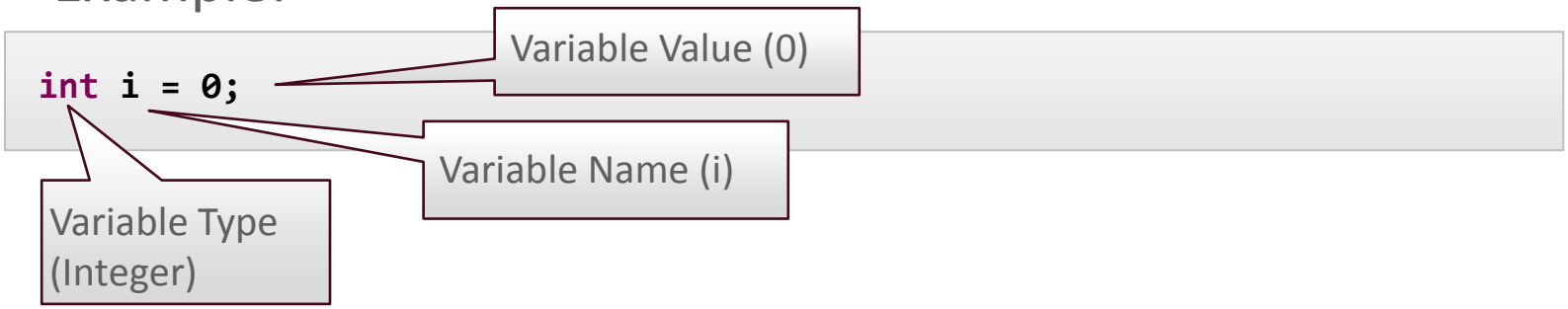
Local Variables

- A local variable is often used within loop constructs.
- While it is similar to a field, it is different because:
  - It is declared inside the method body, not at the beginning of a class.
  - It cannot have a visibility modifier (public or private) in front of its definition.
  - It exists only until the current method finishes running, and is then erased from memory.

A local variable is a variable declared inside the body of the method to temporarily store values, such as references to objects or integers.

Declare a Local Variable

- To create a while loop, first declare the local variable and assign it a value.
- To declare a local variable:
  - Declare the variable type (integer or object reference).
  - Name the variable.
  - Initialize the variable to a number (usually zero).
- Example:



Create the Condition

- Beneath the initialized variable, create the condition that specifies how many times the body of the loop should be executed.
- The loop will continue as long as this condition is true.
- Use a control operator to stop the execution when it reaches the number of executions you specified.

Create the Condition

- Example:
  - Execute the body of the loop while the number of executions is less than, but not equal to, 10.
  - When the loop has been executed 10 times (0-9), it stops.

```
int i = 0;
while (i < 10){
     <code to repeat goes here>
}
```

Insert the Statements to Execute

- In brackets, insert the statements to execute.
- Example:
  - Add 10 new Fly objects with a specific speed and direction.

```
int i = 0;
while (i < 10)
{
     addObject (new Fly (2, 90),150, 100);
}
```

Increment the Loop Variable

- Increment the loop variable as follows:
  - Insert a statement at the end of the loop body to increase the loop variable by 1 each time the loop is executed.
  - Use closed brackets to end the statement.
- This will change the variable with each loop to ensure it does not loop indefinitely.

```
int i = 0;
while (i < 10)
{
     addObject (new Fly (2, 90),150, 100);
     i = i + 1;
}
```

while Loop Example

- When inserted into the BeeWorld constructor it creates 10 Flies when the world 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);

        addObject(new Bee(), 310, 201);
        addObject(new Spider(), 510, 360);

        int i=0;
        while (i < 10) {
            addObject(new Fly(1,90), 505, 70);
            i=i+1;
        }
    }
```

Object Placement and while Loops

- In the previous example, when the constructor is executed, all of the instances are placed at the same coordinates.
- This causes them to sit on top of each other.
- Create an expression that calculates the size of an instance, and then places subsequent instances at different random coordinates.
- The max speed of the flies will be increased within the loop.

Calculate the Placement of Instances

- To program instances to land at different coordinates, replace the fixed x-coordinate for the object's width with an expression that includes:
  - Variable i.
  - Random x and y coordinates.

```
int i=0;
while (i < 10) {
    int xcoord = Greenfoot.getRandomNumber(getWidth());
    int ycoord = Greenfoot.getRandomNumber(getHeight());
    addObject(new Fly(i+1,90), xcoord, ycoord);
    i=i+1;
}
```

Infinite Loops

- If an end to the loop isn't established, the loop keeps executing and never stops.
- Infinite loops are a common problem in programming.
- An infinite loop executes as follows:
  - The variable never changes.
  - The condition always remains true.
  - The loop continues looping forever.

An infinite loop is when the loop keeps executing and does not stop because the end to the loop isn't established.

Animating Objects with a Keyboard Key

- Another way to animate an object is to have the object change the image it displays when a keyboard key is pressed.
- Pseudocode for this action:
  - Switch between four images when a key is pressed.
  - When left key is pressed, show image1.
  - When right key is pressed, show image2
  - When Bee is not turning and current image is image1 –then show image2 else if not turning show image1.
  - Object needs to remember if the Bee is turning - or not.
    - Otherwise, it will rapidly switch the object it displays, and the keyboard key will not be able to control it.

Keyboard Key Example

- The Bee should lean to the left or right when turning.
- Four images are saved in the scenario:
  - One with Bee leaning left, one leaning right, and the last two with the normal flight animation.
- Pseudocode for this action:
  - If Bee is turning then do not use wing movement animation.
  - If turn left show left image, and if turn right show right image.
  - Remember if Bee is currently turning.

Declare Class Variables

- First, add new fields in the Bee class to store the left and right turns, and a field to store the current turn state.

```
public class Bee extends Actor
{
    private GreenfootImage image1;
    private GreenfootImage image2;
    private GreenfootImage imageLeft;
    private GreenfootImage imageRight;
    boolean isTurning;
    int score;
    int lives;
```

Initialize Variables

- Then initialize our class variables in the constructor.

```
/**
 * Bee - Create a Bee and initialize its two images
 */
public Bee() {
    image1 = new GreenfootImage("bee.png");
    image2 = new GreenfootImage("bee2.png");
    imageLeft = new GreenfootImage("beeLeft.png");
    imageRight = new GreenfootImage("beeRight.png");
    setImage(image1);
    score = 0;
    lives = 3;
    isTurning = false;
}
```

Set isTurning Variable

- Next, in our handleMovement() method we wrote earlier we will set the state of our isTurning variable to true if we are turning, else we will set it to false.
- We will also set the appropriate turn image.

```
private void handleMovement() {
    move(1);
    if (Greenfoot.isKeyDown("left")) {
        turn(-2);
        isTurning = true;
        setImage(imageLeft);
    }
    else if (Greenfoot.isKeyDown("right")) {
        turn(2);
        isTurning = true;
        setImage(imageRight);
    }
    else {
        isTurning = false;
    }
}
```

Logic Operators

- To test if Bee is turning when animating :
  - Multiple boolean expressions to express if one or both are true or false.
  - Logic operators to connect the boolean expressions.
  - For example, the statement, "When Bee is not turning and current image is image1..." would be coded as:

```
if (getImage() ==  image1  && !isTurning)
```

Types of Logic Operators

Logic operators can be used to combine multiple boolean expressions into one boolean expression.

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

Logic Operators Example

- Logic operators set the image that appears if the Bee is turning or not.

```
private void animateBee() {
    if (getImage() ==  image1  && !isTurning)
    {
        setImage(image2);
    }
    else if (!isTurning){
        setImage(image1);
    }
}
```

Arrays

- When we created our basic animation we used two images to simulate wings flapping.

- Most animations will have multiple images, but this would increase the number of conditions to test in our code.

Arrays

- With only 4 images our code would look like this:

```
private void animateBee() {
    if (getImage() ==  image1  && !isTurning)
    {
        setImage(image2);
    }
    else if (getImage() ==  image2  && !isTurning)
    {
        setImage(image3);
    }
    else if (getImage() ==  image3  && !isTurning)
    {
        setImage(image4);
    }
    else if (!isTurning){
        setImage(image1);
    }
}
```

- We could easily have 8 or more images!

Arrays

- Using an array, you can hold and access multiple variables from just one variable.

An array is an object that holds multiple variables. An index can be used to access the variables.

How Variables Hold Values

- A simple String variable named "studentname" is a container that holds a value: A single students name .

```
String studentname;
```

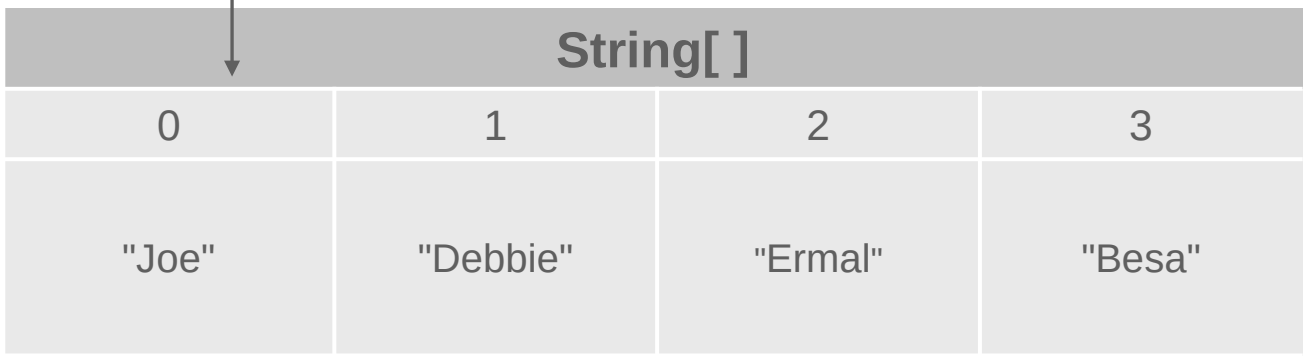
- studentname container example:

Joe

How Arrays Hold Variables

- An array object can hold many variables.
- This array named studentnames can hold many variables.

```
String[] studentnames
```



Variable Declaration for an Array

- To declare an array object, write the variable declaration as follows:
  - Element type:
    - String [ ] for an array of Strings.
    - int [ ] for an array of integers.
  - Square brackets [ ] to indicate that this variable is an array.
  - Variable assignment.
  - Expression that creates the array object and fills it with a number of Strings or integers.

Array Example

- In this array:
  - The studentnames String variable is created.
  - "Joe", "Debbie", "Ermal", and "Besa" are String objects that make up the array object.
  - An array object is assigned to the variable studentnames.

```
String [] studentames;
studentnames = {"Joe", "Debbie", "Ermal", "Besa"};
```

Accessing Elements in an Array

- Use an index to access the elements in the array object.
- To use an index:
  - Each element has an index, starting at position zero [0].
  - Each element's position increases by 1.

Elements are accessed using square brackets [ ] and an index to specify which array element to access. An index is a position number in the array object.

String[ ]				
Index	0	1	2	3
Element	"Joe"	"Debbie"	"Ermal"	"Besa"

Access Elements in an Array

- To access an element in the array, attach the index for that element in square brackets to the array name.
- The statement studentnames[3] accesses the array element at index 3—the String "Besa".
- This is the fourth element in the array.

String[ ]				
Index	0	1	2	3
Element	"Joe"	"Debbie"	"Ermal"	"Besa"

Array Animation

- To better make an instance animate multiple images we could declare an array.
- The array includes all of the images to animate the Bee.
- Declare a field in the Bee class for the array.
- Iterate through the array.

Create the Arrays

- Create an array in the Bee class to store 4 images.
- Create an integer variable to store current image index.

```
public class Bee extends Actor
{
    private GreenfootImage[] images = new GreenfootImage[4];
    private int currentimage;
```

Create the Arrays

- Initialize the new image, the images array and the currentimage variables in the constructor.

```
public Bee() {
    images[0] = new GreenfootImage("bee0.png");
    images[1] = new GreenfootImage("bee1.png");
    images[2] = new GreenfootImage("bee2.png");
    images[3] = new GreenfootImage("bee1.png");
    currentimage = 0;
```

Create the Arrays

- We could use our knowledge of the while loop to better code the previous slide

```
public Bee() {
    images[0] = new GreenfootImage("bee0.png");
    images[1] = new GreenfootImage("bee1.png");
    images[2] = new GreenfootImage("bee2.png");
    images[3] = new GreenfootImage("bee1.png");
    currentimage = 0;
```

- Could be written as

```
public Bee() {
    int i = 0;
    while (i<4) {
        images[i] = new GreenfootImage("bee"+i+".png");
        i++;
    }
    currentimage = 0;
```

Create the Arrays

- Modify the animateBee() method

```
private void animateBee() {
    if (currentimage == 3) {
        currentimage = 0;
    }
    else {
        currentimage++;
    }
    setImage(images[currentimage]);
}
```

Terminology

Key terms used in this lesson included:

- Array
- Elements
- Index
- Infinite loop
- Local variables
- Logic operators
- Loop

Summary

In this lesson, you should have learned how to:

- Create a while loop in a constructor to build a world
- Describe an infinite loop and how to prevent one from occurring
- Use an array to store multiple variables
- Create an expression using logic operators
- Describe the scope of a local variable in a method


ORACLE
ACADEMY