Using Loops, Variables, and Strings

What Will I Learn?

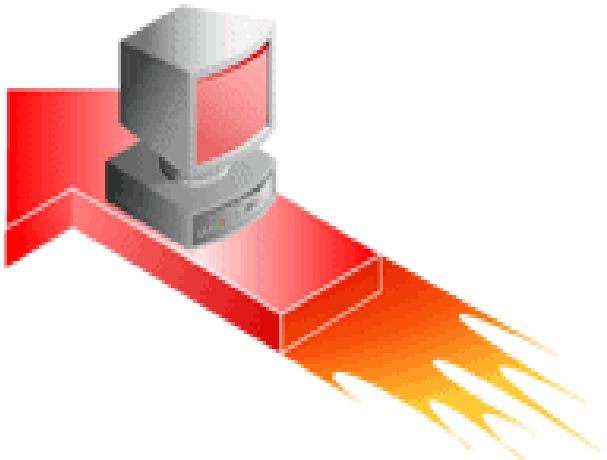
Objectives

- 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
- Use string variables to store and concatenate strings

Why Learn It?

Purpose

You may create games with multiple instances that react to different keyboard commands. Programming games that are more complex, such as a card game for example, can be time-consuming and complicated to program.



In this lesson, you will learn how to use loops and variables: two valuable programming concepts that will save time in the creation of your game.

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.

For example, you can:

- Create 50 instances at once
- Execute a method 10,000 times

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

The components to control how many times the while loop is executed are:

- A loop variable, which is a counter that tells how many times to execute (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 does not 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.

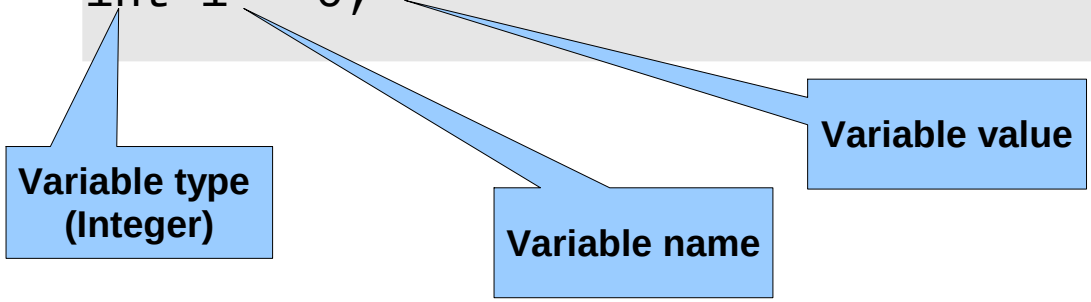
Declare Local Variable

To create a while loop, first declare the local variable and assign it a value. To declare a local variable:

- Declare variable type (integer or object reference)
- Name the variable
- Initialize the variable to a number (usually zero)

Example

```
int i = 0;
```



Create the Condition

Beneath the initialized variable:

- Create the condition that specifies how many times the body of the loop should be executed.
- Use a control operator to stop the execution when it reaches the number of executions you specify.

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){
}
```

Insert the Statements to Execute

In brackets, insert the statements to execute.

Example

Add 10 new Duke objects with a specific keyboard key and sound file attached to each object.

```
int i = 0;
while (i < 10)
{
     addObject (new Duke (“k”, “test.wav”),150, 100);
}
```

Increment the Loop Variable

Then, 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 Duke (“k”, “test.wav”),150, 150);
     i = i + 1;
}
```

While Loop Example

This while loop was inserted into the World constructor and creates 10 Dukes when the world is initialized.

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

/**
 * Write a description of class DukeWorld here.
 *
 * @author Oracle Academy
 * @version JF_S03_L05
 */
public class DukeWorld extends World
{
    /**
     * This constructor creates a new world and the objects that start the
     * game.
     */
    public DukeWorld()
    {
        super(600, 400, 1);
        int i = 0;
        while (i < 10)
        {
            addObject (new Duke("k", "test.wav"), 150, 100);
            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.

We can add an expression that calculates the size of an instance, and then places subsequent instances at different coordinates.

Calculate the Placement of Instances

To program instances so they land at different coordinates and not on top of each other, replace the fixed x-coordinate for the object's width with an expression that includes:

- Variable `i`
- Multiplication operator (*)
- Fixed offset integer so the first object does not land off-screen

```
public class DukeWorld extends World
{
    /**
     * This constructor creates a new world and the objects that start the
     * game.
     */
    public DukeWorld()
    {
        super(600, 400, 1);
        int i = 0;
        while (i < 10)
        {
            addObject (new Duke("k", "test.wav"), i*150 + 30, 100);
            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 is when the loop keeps executing and does not stop because the end to the loop isn't established.

With an infinite loop:

- The variable would never change.
- The condition would always remain true.
- The loop would continue looping forever.

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 two images when a key is pressed
- When key is pressed, show image1
- When key is not pressed, show image2
- Object needs to remember if the key is pressed or not (otherwise, it will rapidly switch the object it displays with no control by the keyboard key)

Keyboard Key Example

Duke's arm should wave if a keyboard key is pressed. Two images are saved in the scenario: One with Duke's arm up, and one with his arm down.

Pseudocode for this action:

If Duke's arm is up, and the keyboard key is down

then

change the image to show the image with Duke's arm down.

Remember that Duke's arm is currently down.

Specify Image to Display

First, write the code in the class's act method to specify the image to show if the key is pressed down, or not. Use the following methods:

- `isKeyDown` Greenfoot method (use dot notation)
- `setImage` method

```
public void act()
{
     if (Greenfoot.isKeyDown(“k”)){
        setImage (“Duke.PNG”);
     }
     else {
        setImage (“DukeDown.PNG”);
     }
}
```

Declare isDown Variable

Next, declare the `isDown` variable in the class's source code, and assigned it to Duke, to tell Duke to remember if the key is pressed down or not.

True/false condition:

- True when keyboard key is pressed down
- False when keyboard key is not pressed down

isDown Variable Example

Variable `isDown` is declared and set to false, because the scenario starts with the keyboard key not down.

```
public class Duke extends Animal
{
    private GreenfootImage image1;
    private GreenfootImage image2;
    private boolean isDown;
    private String key;
    private String sound;
    /**
     * Create a Duke and initialize his two images. Link Duke to a specific keyboard
     * key and sound.
     */
    public Duke(String keyName, String soundFile)
    {
        key = keyName;
        sound = soundFile;
        image1 = new GreenfootImage("Duke.PNG");
        image2 = new GreenfootImage("DukeDown.PNG");
        setImage(image1);
        isDown = false;
    }
```

Logic Operators

To test if Duke's arm is up or down when a key is pressed, this requires:

- Multiple boolean expressions to express if one or both are true or false.
- Logic operators to connect the boolean expressions.

For example, the first statement:

If Duke's arm is not down, and the “d” is down...

would be coded as:

```
if (!isDown && Greenfoot.isKeyDown(“d”) )
```

Types of Logic Operators

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

Logic Operators

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 “d” key is up or down.

```
/**
 * Act - do whatever the Duke wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(2);
    if (!isDown && Greenfoot.isKeyDown("d"))
    {
        setImage ("Duke.PNG");
        isDown = true;
    }
    if (isDown && !Greenfoot.isKeyDown("d"))
    {
        setImage ("DukeDown.PNG");
        isDown = false;
    }
    lookForCode();
}
```

Play Sound

Now that the statement is programmed to animate Duke, the last step is to program the statement for Duke to make a sound when the “d” key is pressed, in addition to moving his arm.

Define the method to play the sound, so you can call it in the act method when the specific key is pressed down.

Define Play Method in Class

First, define a method in the class called `play`. Write the method below the act method, as shown below.

This method:

- Calls the `playSound` method from the Greenfoot class using dot notation in the body of the if statement.
- Includes the sound file name to play.

```
/**
 * Play a sound.
 */
public void play()
{
    if (Greenfoot.isKeyDown("d"))
    {
        Greenfoot.playSound("test.wav");
    }
}
```

Enter Play Method in Act Method

Enter the `play` method in the act method:

- Enter it into one of the if statements to have it play when a keyboard key is pressed.
- Enter it below the if statement to have it play continuously during the game.

```
/**
 * Act - do whatever the Duke wants to do. This method is called whenever
 * the 'Act' or 'Run' button gets pressed in the environment.
 */
public void act()
{
    move(2);
    if (!isDown && Greenfoot.isKeyDown("d"))
    {
        setImage ("Duke.PNG");
        play();
        isDown = true;
    }
    if (isDown && !Greenfoot.isKeyDown("d"))
    {
        setImage ("DukeDown.PNG");
        isDown = false;
    }
    lookForCode();
}
```

Arrays

When you create multiple instances using a while loop constructor, each receives the same sound file and keyboard key assignment.

In most situations, this isn't ideal. Instances may need to react to different keyboard keys, or play different sounds.

Using an array, you can hold and access multiple variables, and assign different values to new instances each time they are created.

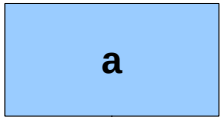
An array is an object that holds multiple variables. An index can be used to access the variables.

How Variables Hold Values

A simple `String` variable named “keyname” is a container that holds a value: A single keyboard key's name.

```
String keyname;
```

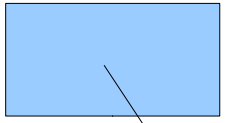
Keyname container



How Arrays Hold Variables

An array object can hold many variables. This array named `keynames` can hold many variables.

`String [ ] keynames`



String[ ]			
0	1	2	3
“a”	“s”	“d”	“f”

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 an unlimited number of Strings or integers

Array Example

In this array:

- The keynames string variable is created
- “a”, “s”, “d”, and “f”, Strings are the array object
- Array object is assigned to the variable keynames

```
String [] keynames;
keynames = {“a”, “s”, “d”, “f”};
```

Accessing Elements in an Array

Use an index to access the elements in the array object.

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.

To use an index:

- Each String or integer has an index, starting at position zero [0].
- Each element's position increases by 1.

	String[ ]			
Index	0	1	2	3
Element	“a”	“s”	“d”	“f”

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 `keynames[3]` accesses the array element at index 3—the String “f”. This is the fourth element in the array.

Play Different Sounds When Keys Pressed

To make an instance able to play a variety of sounds based on which keyboard key is pressed:

- Create two arrays in the World class that include:
  - Names of keyboard keys for Duke instances
  - Names of sound files for Duke instances
- Declare fields in the World class for those arrays
- Store the filled arrays

Create the Arrays

Create two arrays are in the World class, listing the key names and sound files to use.

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

/**
 * Write a description of class DukeWorld here.
 *
 * @author Oracle Academy
 * @version JF_S03_L05
 */
public class DukeWorld extends World
{
    private String[] keyNames =
        {"a", "s", "d", "f"};
    private String[] soundNames =
        {"test.wav", "testl.wav", "test2.wav", "test3.wav"};
    /**
     * This constructor creates a new world and the objects that start the
     * game.
     */
    public DukeWorld()
    {
```

Create makeDukes Method

For clarity in coding, we moved the creation of Duke into a separate method called `makeDukes`.

```
public class DukeWorld extends World
{
    private String[] keyNames =
        {"a", "s", "d", "f"};
    private String[] soundNames =
        {"test.wav", "testl.wav", "test2.wav", "test3.wav"};
    /**
     * This constructor creates a new world and the objects that start the
     * game.
     */
    public DukeWorld()
    {
        super(600, 400, 1);
    }
    /**
     * This constructor creates Duke and assigns the keys ar  d file names.
     */
    private void makeDukes()
    {
        int i = 0;
        while (i < 4)
        {
            Duke duke = new Duke(keyNames[i], soundNames[i]);
            addObject (duke, i*150 + 30, 100);
            i = i + 1;
        }
    }
}
```

Duke object assigned to a local variable called duke.

The loop variable `i` accesses all of the key strings and sound file names in the array object, in the order entered.

Call the makeDukes Method

Finally, call the `makeDukes` method in the World constructor.

```
public class DukeWorld extends World
{
    private String[] keyNames =
        {"a", "s", "d", "f"};
    private String[] soundNames =
        {"test.wav", "test1.wav", "test2.wav", "test3.wav"};
    /**
     * This constructor creates a new world and the objects that start the
     * game.
     */
    public DukeWorld()
    {
        super(600, 400, 1);
    }
    /**
     * This constructor creates Duke and assigns the keys and sound file names.
     */
    private void makeDukes()
    {
        int i = 0;
        while (i < 4)
        {
            Duke duke = new Duke(keyNames[i], soundNames[i]);
            addObject (duke, i*150 + 30, 100);
            i = i + 1;
        }
    }
}
```

String Concatenation

To reduce the number of redundant characters or phrases you need to type into each array, use string concatenation.

String concatenation combines two Strings together into one. It is represented by a plus symbol (+).

A plus symbol between two strings puts them together into a single string.

String Concatenation Example

Instead of entering ".wav" after each sound file name in the array, add + ".wav" after the soundName value in the constructor.

```
Duke duke = new Duke(keyNames[i], soundNames[i] +
".wav");
```

The name stored in the array—test—combined with ".wav" produces the full file name that the program understands, "test.wav".

Terminology

Key terms used in this lesson included:

Array

Elements

Index

Infinite loop

Local variables

Logic operators

Loop

Summary

In this lesson, you 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 used to create a world
- Create an expression using logic operators
- Describe the scope of a local variable in a method
- Use string variables to store and concatenate strings

Practice

The exercises for this lesson cover the following topics:

- Creating a world constructor method with sound and keyboard input
- Using loops and arrays
- Journaling while loops and arrays