**2.1.1 Autumn – Object Interactions**

**Introduction**

In this assignment we start to go deeper into object-oriented programming by creating objects that can call methods from other objects to create an interactive world. This assignment will explore the power of OO programming and all the fundamentals we have learned. Our goal in this assignment is to learn communication between objects. This time we will use arrays of objects rather than arrays of strings or integers.

Concepts covered are:

* Object references
* Reference vs value
* get methods
* Client class
* Keywords: **.class**

**Materials**

* Computer with Greenfoot

**Object References**

1. Download autumn.zip and unzip into your project folder. Open and run the scenario – it’s quite basic. Look at the source code and see if you can determine how it works.
2. We are used to variables storing **values** inside them. For example, what are the values of x and yafter the following lines of code are executed?

int x = 5;

int y = 1;

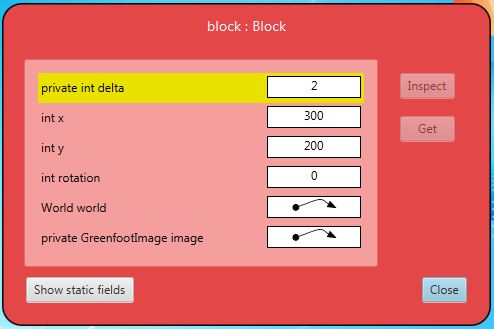
x = y;

y = y + 1;

When primitive datum are assigned to variables the value is stored inside a ‘box’ with the variable name written on the outside. When the value stored in y (1) is assigned to x it replaces the old value of x. In line four we replace the old value of y with y + 1 (2). This action does not alter whatever value is stored inside of x.

In other words, the primitive values stored in variables are totally independent of one another. This is not the case with objects.

1. Objects, in contrast, are complicated data that contain variables, values, and methods inside of them. When we try to store an object in a simple variable, a **reference** is stored rather than a value or the object itself. In Greenfoot a reference looks like an arrow when you inspect a variable or object. Right click on the Block object on the stage and this is what you see:



You can see that the Block object has six variables stored inside of it: delta, x, y, rotation, world, and GreenfootImage.image. The first four (delta, x, y, rotation) are storing primitive **values**, the last two (world, and GreenfootImage.image) are storing **references**.

1. A reference simply allows one object to know the memory location and methods stored inside another different object without storing the actual object. For example: block (this object) can use its two references to call methods from world and image. An example we have seen before is using the world variable **reference** to call its addObject() method:

world.addObject(new Leaf(),100,100);

What’s really going on here is that the block object is “talking” to the world using its reference.

The obvious question here is: how do we get this reference in the first place? The answer is that we generally use **get methods.** These are methods that begin with the word ‘get’ and return object references.

For example, the getWorld() method of the Actor class will obtain a reference for us. We’ve seen this in other projects with code like the lines below:

World world = getWorld();

world.addObject(new Leaf(),100,100);

|  |
| --- |
| * Modify the code in the Block class to add new leaves to the world whenever the block hits the edge of the world. Add the leaves at the location of the block when it hits the edge.   + We will add the code to the checkEdge() method of the Block class (starting at line 24).   + Add the two lines shown above to the method.   + You can also find methods in the Actor API that return the current location of the block; use this to create leaves in the same spot as the block.   + Test the code (you may want to increase the speed of the program). |

1. Because we are calling world methods inside the Block class object, we call the Block class a **client class** of the World class. That is: Any class that calls methods from another class is a client class to that class.

Let’s practice using references by programming the leaves that are in the world to turn when hit by the block. The question is: how do we get the reference for a randomly created leaf? We need a new method to do this. We can use Greenfoot’s *collision detection* methods. These are methods that detect if an actor touches another actor.

|  |
| --- |
| * We want the leaves that are touched by the block to turn a random number of degrees. We will use a collision detection method called getOneIntersectingObject() in the Actor class.   + Open the Actor class API and find the getOneIntersectingObject() method. Read the description.   + Create a new method in the Block class called checkLeaf().   + Create an if statement in the new method that checks if the block is touching any object of class Leaf. Hint: Adding **.class** to the end of a class name means “any object of that class”. Look at the Actor API if you need help.   + Add this line of code inside the if block:   Leaf leaf = (Leaf)getOneIntersectingObject(Leaf**.class**);   * + Now that we have the reference stored in the variable leaf, call the turn() method of the Actor class on the leaf and turn it a random number of degrees. You can use the Greenfoot class’ getRandomNumber() method to generate a number of degrees.   + Call the new method in the act() method of the Block class.   + Test your code. |

1. Look at the lines we just added and the API. Do you understand how these lines work? Let’s dissect the first line.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Variable type** | **Variable name** |  | **Cast as a Leaf** | **This method obtains the object reference we want** | **Argument – any object of the Leaf class** |
| Leaf | leaf | = | (Leaf) | getOneIntersectingObject | (Leaf**.class**); |

This line declares a new variable and stores a reference to any leaf it grabs in the variable. We need to cast as a Leaf because the method returns an object of class Actor but we want class Leaf.

1. Know that we know about references, let’s consider the code below and compare it with the code in #2 above:

World thisWorld = getWorld();

thisWorld.setBackground(“autumn.jpg”);

World thatWorld = thisWorld;

thatWorld.setBackground(“winter.jpg”);

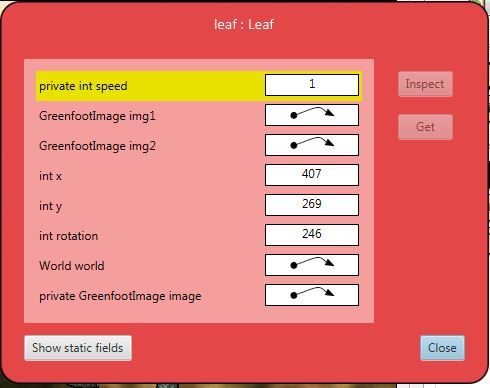
What is the background of thisWorld? If references worked the same as values, the background would be autumn.jpg … but it isn’t. Instead, **when a variable storing a reference is assigned to a new variable, they will both store the same reference.** So thisWorld and thatWorld both refer to the same place, like using a name and a nickname for the same person.

1. Now that we understand **references** let’s improve our autumn program:

|  |
| --- |
| * In the checkLeaf() method, make the leaves rotate between 90 and -90 degrees when hit by the block. * Add some code to the setup() method of the world that will randomly place from 5-18 leaves in the world using a for loop. * Create a new Actor subclass called Apple and give it the apple image. Modify the setup() method to place 6-19 apples in random locations in the world. * Create yet another Actor subclass called Pear with a suitable image and place 1-8 in the world using setup(). * Add code to the Block class that moves pears to the right when hit by the block. If the pear is on the right edge, have it appear at the left edge. |

**Conclusion**

1. Describe the difference between a value and a reference. Give 2 examples (each) of when we might store a value in a variable and when we might store a reference.
2. Which of the following variables in this object are references and which are values?



1. What is a get method? What are two examples used in this assignment?
2. When should you use **.class**? When shouldn’t you use **.class**?
3. Examine the code below:

public class SomeClass extends ThisClass

{

int x = 0;

int y = 1;

public void setX()

{

OtherClass a = OtherClass.getMe();

}

}

1. Which class is a client class of SomeClass? Which line of code tells you?
2. (review) Which class is a parent class? Which line of code tells you?
3. Examine the code below. The method getObject() returns a reference to an object of ObjectA and the method setX() changes the value of the public variable x in an ObjectA object to the integer given as an argument.

//Method implementations not shown

ObjectA myObject = getObject();

myObject.setX(1);

ObjectA thisObject = myObject;

thisObject.setX(2);

What value of x is stored in myObject? What about thisObject? Why?