**1.1.2 Introducing Greenfoot - Object Oriented Programming**

**Introduction**

**Greenfoot** is an interactive Java development environment designed primarily for educational purposes at the high school and undergraduate level. It allows easy development of two-dimensional graphical applications, such as simulations and interactive games. Greenfoot teaches **object oriented (OO)** programming in Java. Create 'actors' which live in 'worlds' to build games, simulations, and other graphical programs.

We will cover:

* Object-Oriented programming
* IDE
* compile
* class diagram
* class
* subclass
* superclass
* inheritance
* object (instance)
* instantiation
* method
* invoke
* return
* parameter
* arguement
* keyword: **void**

**Materials**

* Computer with Greenfoot
* Source files Part I & II: Leaves & Wombats

**Activity**

To begin learning Java, you will use an **Integrated Development Environment**, or **IDE**, called **Greenfoot**. An IDE does a lot of work for you:

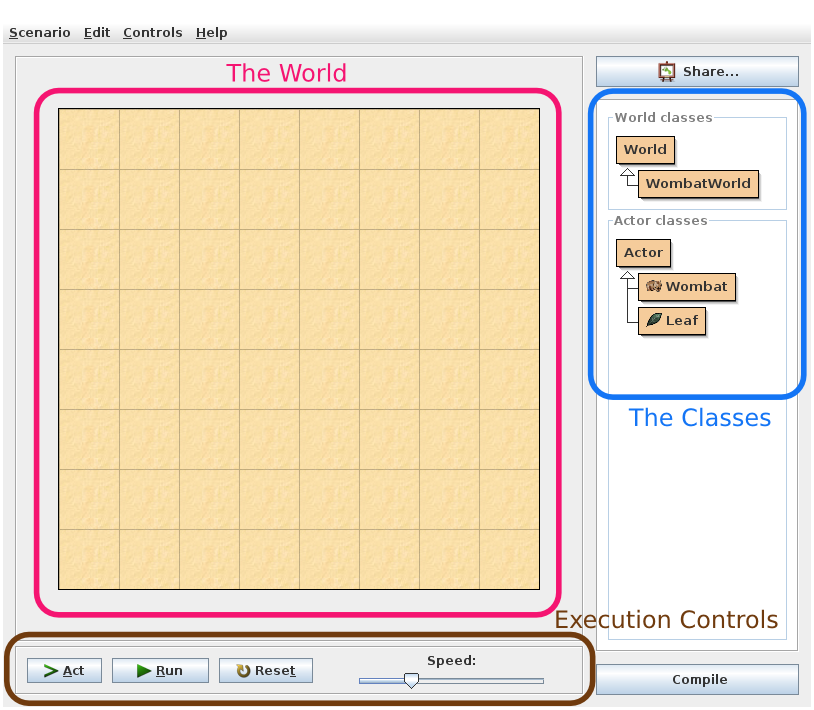
* Provides an editor to edit the files in your project.
* Organizes project files.
* **Compiles** your code, converting it to a machine language the computer can process.
* Runs your program.

**Part I: Intro to Classes, Objects, & Methods**

|  |
| --- |
| 1. Create a folder on your desktop called “Greenfoot Projects”. Download leaves-and-wombats.zip and then extract the folder into your Greenfoot Projects folder. 2. Open the Wombats scenario in Greenfoot; If you don't see the world, make sure you unzipped the file folder - you can’t work with zipped files. If the boxes on the right are gray and have diagonal slashes over them then double-click the box that is grey and click on the "**Compile**" button in the top left. Do this until none of the boxes are grey. |

1. You should be looking at the Greenfoot **IDE** (Integrated Development Environment). IDEs are a collection of programs all in one place and are used by programmers (developers) to make software. The Greenfoot IDE consists of 3 sections: the world, the class diagram and the execution controls.
   1. The world is the largest area. It is where the program will run.
   2. The class diagram shows Java classes and how they are related - more on this in a bit.
   3. The execution controls are used to control your program.

Here’s the labeled IDE - you should see this:



1. Let’s look at the class diagram first. Java is an **object-oriented (OO)** language and the idea of objects and classes is fundamental to how it works. OO programs consist of a collection of interrelated classes that are used to make objects. A **class diagram** is a diagram of all of the classes in an OO program and their relationships. For example, there are two main **classes** in this program - World and Actor. **Classes** are blueprints that are used to make objects for the program. **Objects** (also called **instances**) are the various items in the program that have properties and behaviors.
   1. A good analogy to use is this: **classes** are like cake molds while **objects** are the cakes themselves.
   2. We can also modify the basic cake mold to make an improved (or more specific) cake mold called a **subclass**.

WombatWorld, Wombat, and Leaf are subclasses and in the class diagram they point to the class they came from. Subclasses can do whatever their **superclass** (the class they were made from) can do but have slightly altered behavior. For example: Wombat is a subclass of Actor - it can do everything that Actor can do plus it has a few tricks of its own. We say that Wombat **inherited** its traits from Actor.

WombatWorld is a subclass - what is its superclass?

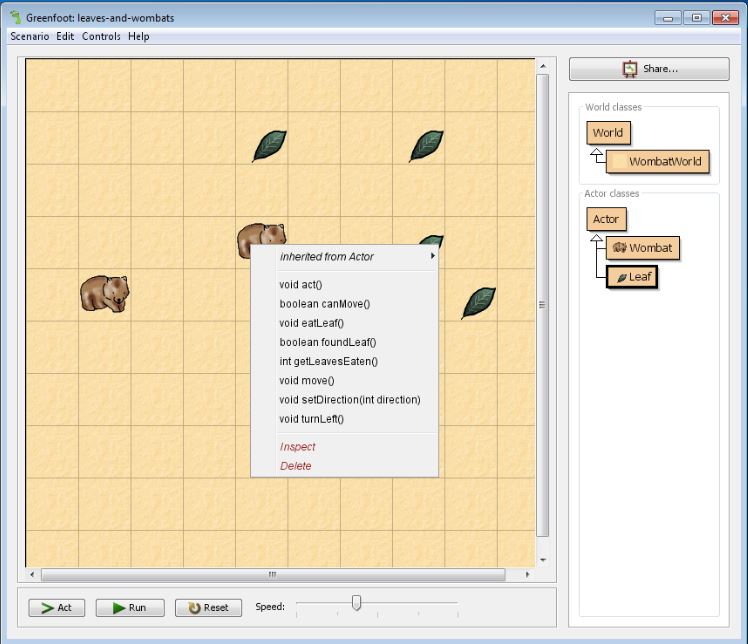
1. We will now place some **objects** into the world. Objects are created from classes, like a cake from a cake mold. Right-click the Wombat class in the class display. You will see a pop-up menu like this:

|  |  |  |
| --- | --- | --- |
| C:\Users\anthony pittman\Desktop\class-popup-Wombat-plain.png | 1. Look at the newWombat() option: this is an example of a procedure or function in Java; we call them **methods**. Think of them as the actions an object can take.  |  | | --- | | Choose 'new Wombat()' from the menu. Then click anywhere in the world. You have just created a wombat (in Java terms: an object) and placed it into the world. Making a new object from a class is called **instantiation**. | |

|  |
| --- |
| 1. Wombats eat leaves, so let's instantiate some leaves. Right-click the Leaf class, select 'new Leaf()' and place the leaf into the world. Place many leaves in the world. |

There is a shortcut to place several objects a bit faster: shift-clicking into the world. Make sure the Leaf class is selected (left click on it in the classes panel, and it will get a thicker black border), then hold down the Shift key and left-click in the world several times. You will place one object of the selected class at every click.

1. Right click a wombat that you just placed. You will see a list of methods associated with the wombat. If you choose a method, the wombat will perform that action (we call this **invoking** a method).



|  |
| --- |
| 1. **Invoke** the void move() method from the list. What does this method do? |

1. You will notice that methods are preceded by a keyword like **void**, **boolean, int,** or **new**. These are called **return** types. A return of **void** means that the object simply performs an action and does not give any information back to you. A return of any other type means the object will give you information rather than performing an action. The keyword specifies the type of information - more on this in future lessons.

|  |
| --- |
| Try out a the method getLeavesEaten() so you can see a method that does not have a void return. Try out a few methods. |

1. You may have noticed that some methods ask you for input - for example: void setDirection(int direction). This method requires a parameter. A **parameter** is an additional bit of data that a method requires in order to do its job. Parameters are listed inside the ( ) after the method name. The keyword before the parameter tells you which type of data you should provide - more on that later. Suffice it to say that **int** means it needs a whole number. The specific number we type in when we invoke the method is called an **argument**.

|  |
| --- |
| Try out setDirection() and enter ‘2’ in the input box when asked. What happens? Try some other values. |

|  |  |  |
| --- | --- | --- |
| **Method Signature** | | |
| **return type** | **method name** | **parameter(s)** |
| **void** | setDirection | (**int** direction) |

Invoking a method:

|  |  |
| --- | --- |
| **method call** | **argument** |
| setDirection | (2) |

|  |
| --- |
| 1. Do the following:    1. Place 2 wombats in the world and make them face each other    2. Get the getLeavesEaten() method to return 2 on one of your wombats.    3. Place a wombat and a good number of leaves in the world. Invoke the wombat’s act() method    4. The world itself is also an object. Right click on the world and you will get its methods. Try out the two methods. |

**Part II: Greenfoot Execution Controls**

1. Take a look at the execution controls at the bottom of the main Greenfoot page.

|  |
| --- |
| With several wombats and leaves in the world, press the *Act* button. Each object now acts -- that is: each object does whatever it is programmed to do. In our example, leaves are programmed to do nothing, while wombats are programmed to move forward. Wombats also like to eat leaves. If they happen to come across a leaf in their path, they will eat it.   * 1. What is the difference between invoking the act() method and pressing the *Act* button? Compare with several wombats in the world.   2. The wombat has three actions that it can do - what are they? In what order does it do them (if a leaf is present)?  1. Click the *Run* button. What does this button do? 2. You will notice that the *Run* button changes to a *Pause* button; what happens when you press this button? 3. The slider next to the Act and Run buttons sets the speed. Click Run and then change the slider, and you'll see the difference. |

1. If you have many objects in the world that you do not want anymore, and you want to start all over, there is one easy option: throw away the world and create a new one. This is usually done by clicking the *Reset* button in the execution controls. You will get a new, empty world. The old world is discarded (and with it all the objects that were in it) -- you can only have one world active at a time.

|  |
| --- |
| Reset your world and make a new one. |

1. Take some time to experiment with the world.

**Conclusion**

1. What is an IDE? What are the three parts of the Greenfoot IDE?
2. Look at the following class diagram:

|  |  |
| --- | --- |
|  | * 1. What is the superclass that Space belongs to?   2. What superclasses does Bullet belong to?   3. What are the three subclasses of Mover?   4. Which classes have no superclass?   5. What classes are inheriting the methods of the Actor class? |

1. Take a look at the following methods:

void move()

void setLocation(double x, double y)

int getStability()

void createStars(int number)

* 1. Which methods return something?
  2. Which methods have parameters, how many, and what are they?
  3. Which of these are like asking questions and which are like giving commands? Why?
  4. If I type setLocation(2.5, 3.5) what are my arguments?

1. What does the act() method do in Greenfoot? What does the *Act* button do? How do they differ?
2. What does the *Run* button do in Greenfoot?