**Chapter 1 – Leaves and Wombats**

Vocabulary

1. Objects: Individual copies of a class that create elements in the world that we can work with
2. class: The blueprint for a specific type of object. The class can have copies made of it that actually become working Objects in the game.
3. Methods: operations in Java that perform commands
4. Return type: the word at the beginning that specifies what a method call will return when invoked
5. Void: when a method does not return a value or any information
6. Commands: methods with void return types
7. Questions: methods with non-void return types
8. Parameter: a method to pass in additional data to a method
9. Types: parameters and return values have this (ex. Boolean for true/false)
10. Signature: the description of each method shown in the object menu
11. Subclass: a class that represents a specialization of another.
12. Source code: the code that defines what objects of this class can do
13. Compilation: the translating of source code to machine code

Exercises

1. I created 2 wombats and 4 leaves in the world on different squares.
2. The move() method moves the wombat forward one square in the direction that it is currently facing. Invoking the move() method multiple times causes the wombat to move the number of times move() is invoked; if the move() method is invoked when the wombat is facing the edge of the screen, then it will not move. The turnLeft() method causes the wombat to rotate counter-clockwise 90 degrees.
3. I invoked the canMove() method on my wombat, in which it would return false if it was facing the edge. It would return true if its body was parallel to an edge.
4. I tried the int getLeavesEaten() method on one of my wombats, and it had a return value of 0, meaning it had not eaten any leaves yet, because it was newly created. If I let it eat a leaf, the return value should be 1.
5. I invoked the setDirection(int direction) method; When I inputted an integer greater than 3, nothing happened to the wombat. When I inputted a non-whole number, it said there was an error.
6. I invoked a wombat’s act() method several times. When it was facing the edge, it would turn direction so that it would be parallel to the edge. When the wombat was sitting on a leaf and i invoked its act() method, it would eat the leaf.
7. The Act button near the bottom of the Greenfoot window makes it so that everytime I press it, it invokes the act() method for both wombats.
8. The Act button invokes all wombats as if the act() method for all wombats were triggered at the same time, moving all wombats at once. The act() method only works on one wombat at a time, so it’ll only move one wombat at a time.
9. The Run button puts the wombats on a continuous cycle, as if the Act button was being held down continuously.
10. We placed a rocket into the world and invoked the setGunReloadTime method to 5, which caused the firing speed to be much faster than what it was before.
11. We invoked the getShotsFired() method and it returned the value 69, meaning we fired 69 shots. This was the fewest number of shots we fired.
12. When we invoked getSpeed() method, it returned that the initial speed of the rocket was 3.
13. When we invoked getStability() method, the asteroid returned a stability value of 64.
14. We invoked setSize(int size) and set the size of the asteroid to 128, which made a very big asteroid.
15. We changed the source of the rocket, adjusting the gun reload time to be 5, which made firing much more efficient.

**Chapter 2 – The First Program: Little Crab**

Vocabulary

1. Method call: an instruction that tells an object to perform an action through invoking a method
2. Parameter: an additional bit of information that needs to be provided when calling some methods
3. In sequence: the way that multiple instructions are executed, one after the other, in the order in which they are written
4. Error message: what is displayed when the compiler finds an error
5. Inherits: the act of a subclass containing or using all methods that its superclass defines
6. Void return type: will issue a command when used to call a method
7. Non-void return type: Will ask a question when used to call a method
8. If-statement: makes it possible to execute commands only if some condition is true

Exercises

1. I placed a crab into the world and pressed the Run button, and nothing happened.
2. I changed the act method in my crab class to include the move() instruction. When I press the Act button, the crab will move moves a little to the right. If I press Run, the crab keeps moving until it hits the edge.
3. I placed 5 crabs into the world and let the scenario run. All the crabs moved to the right until they hit the edge.
4. I replaced move() with turn(5), in which the crab turns continuously in place. I changed the value to 10, making the crab spin even faster.
5. To make the crab turn left, I inputted a negative value in turn().
6. I used a move() and turn(5) instruction in my crab’s act method. The crab doesn’t turn in place, but rather, spins while forming a circular movement. I changed the input value inside turn() to 10, which made the cran turn faster.
7. I removed the semicolon after move() and pressed compile. It showed my error (“;” expected), meaning it’s expecting a semicolon to be placed. When I misspelled move() to moe(), the error displayed was that the compiler could not find the symbol moe().
8. I also deleted an open bracket, in which it showed there was an invalid method declaration and that it was in need of a return type. I also removed an input value in turn(), and the error was that no suitable method found for turn (no arguments). I replaced a semicolon with a period and the compiler was unable to recognize the symbol and provoked the message <identifier> expected. I replaced the input value inside turn() to an asterisk, and the error message provoked was that it was an illegal start of expression. I misspelled turn() to turn() and it said that it could not find the symbol.
9. I opened the editor for the Animal class and looked at Documentation view. Under the “Method Summary” section, it shows this class contains 6 methods.
10. I called the boolean atWorldEdge() method on a crab and it returned false.
11. I moved the crab to the edge of the screen then called the atWorldEdge() method, and it returned true.
12. I typed the code discussed and whenever the crab runs into the edge, it turns.
13. I changed the input of the turn method to be a negative value, which makes the crab turn in the opposite direction whenever it hits the edge, as compared to the previous exercise.
14. I placed the move() statement inside the if-statement. When the crab is anywhere besides at the edge, it won’t move. When it is placed at the edge, it will do one turn and just stop.

**Chapter 3 – Improving the Crab (more sophisticated programming)**

Vocabulary

1. Dot notation: when a method wished to call is not in our own class or or inherited, need to specify the class or object that has the method before the method name, and is followed by a dot.
2. Static: methods that belong to class itself are marked with static at the beginning of the method signature
3. Class methods: methods that belong to classes that are marked with static at the beginning of the method signature
4. Method definition: defines a new action for objects of the class, and will be executed later when a method is called with a method call
5. Comments: what is written for human readers to explain the purpose of what the method is; the comment is ignored by the computer
6. Save A Copy As: helps to keep a copy of interim stages of scenarios
7. API documentation: shows all classes and methods available in Greenfoot

Exercises

1. getRandomNumber(10) < 1
2. getRandomNumber(100) < 7
3. I tried experimenting with random course changes to get different probabilities for turning.
4. I modified the code so that the crab would turn a random amount between 0 and 44, so it turns a different amount each time that it does turn.
5. I changed the code to turn(Greenfoot.getRandomNumber(90)-45); so that the crab turns either left or right by up to 45 degrees each time that it turns.
6. I ran the scenario with multiple crabs in the world and then all turn independently.
7. I added some crabs and worms to the world and ran the scenario. The crabs move and turn how they would, and the worms do nothing but stay in place. If a crab runs over a worm, nothing happens to either animal.
8. I made a new method named randomTurn nad moved the code that does the random turning from the act method to the new randomTurn method.
9. I made a new method named turnAtEdge nad moved the code that does the random turning from the act method to the new turnAtEdge method.
10. I added a new subclass of Animal, called Lobster.
11. When I place lobsters into the world and run the scenario, they don’t do anything.
12. I copied the complete act method from the Crab class into the Lobster class, and copied the complete lookForWorm, turnAtEdge, and randomTurn methods.
13. I changed the lobster code so that it looks for crabs instead of worms, and changed the name lookForWorm to lookForCrab.
14. I placed a crab, three lobsters, and eight worms into the world and ran the scenario. The crab does not manage to eat all the worms before it is caught by a lobster.
15. I removed the random turning code from the crab.
16. I removed the code from the crab that does the turn at the edge of the world.
17. I added the code into the crab’s act method that makes the crab turn left whenever the left cursor key is pressed and tested it.
18. I added a code into the crab’s act methot so that the crab will turn right whenever the right cursor key is pressed down.
19. I made a new method called checkKeypress that includes the codes that check key-presses and does the turning.
20. I opened the Greenfoot API, found the Method Summary section, and found the method called stop() that pauses the execution of running the scenario.
21. The stop() method does not expect any parameters and its return type is void.
22. I added the Greenfoot.stop() code under the lookForCrab method in which when a lobster eats a crab, the game stops.
23. From the Greenfoot Class Documentation, I found the method playSound that will play sound from a file. It expects the name of a string soundfile from java.
24. I added the playSound code under the crab’s lookForWorm method so that when a crab eats a worm, it will play a slurp sound. I also added a code in the lobster’s lookForCrab method so that when it eats a crab, it makes an au sound.
25. I did not make my own sounds for when the crabs eat or for when the lobsters eat.

**Chapter 4 - Finishing the crab game**

Vocabulary

1. Constructor: special kind of method that is executed automatically whenever a new instance is created
2. New: helps to programmatically create java objects using this new keyboard
3. GreenfootImage: Greenfoot actors maintain their visible image by holding an object of type
4. Instance variables (fields): used to store information for later use
5. Assignment statement: assigns an object or a value to a variable
6. Reference: When an object is assigned to a variable, the variable contains a reference to that object
7. Equal: == compares one value with another; the result is either true or false
8. If/else statement: executes a segment of code when a given condition is true, and a different segment of code when it is false

Exercises

1. I added code to the CrabWorld constructor to create a crab automatically.
2. I added code to the CrabWorld constructor to create three crabs automatically.
3. I added code to the CrabWorld constructor to create 10 worms automatically.
4. I moved all the code that creates the objects into a method called populateWorld in the CrabWorld class. I declared the populateWorld method.
5. I replaced the coordinate values of the worms by using the Greenfoot.getRandomNumber() code.
6. The two methods that change an actor’s image are setImage and getImage. Their parameters needed are the name of a string file from java. The return nothing.
7. I selected Inspect from the crab’s popup menu and noted the variables that are shown in the crab object: int x, int y, int rotation, World world, private GreenfootImage image.
8. The crab has to have variables when placed into the world, since it inherits the variables from the Actor class.
9. I added the variable declarations shown on Code 4.2 to my Crab class.
10. I inspected the crab object again. The variables and their values now are [private GreenfootImage image1 (null), private GrennfootImage image2 (null), int x (150), int y (100), int rotation (0), World world, private GreenfootImage image.
11. I added the constructor to my Crab class.
12. The only thing that’s changed when I inspected the variables now in the crab object is the values of the private GreenfootImage image1 and image2 are not null. They have arrows as their values now.
13. I added the 4.4 code into my Crab class.
14. I created a new method called switchImage, and added my image switching code and called this method.
15. I called the switchImage method interactively from the crab’s popup menu and it works.
16. I added the code so that once the crab eats 8 worms, the game stops, and the fanfare sound plays.
17. I opened the object inspector for my crab object before I ran the game it it showed my crab had eaten 0 worms. I let the inspector open as I played, and every time the crab it a worm, the wormsEaten variable increased by 1.
18. I added a counter that gets incremented in the act method. Every Time it reaches 3, the image changes and the counter is reset to 0.