**1.1.6 Introduction to Strings & Output**

**Introduction**

Moving on to more advanced topics than basic syntax - we will look at the Java String object, learn to manipulate it to create outputs, and add to the complexity of our knowledge.

Concepts covered are:

* Code flow
* Method chaining
* Output
* String
* Index
* String operations
* Concatenation
* Private methods (more details; first covered in 1.1.4)
* Keywords: **this**

**Materials**

* Computer with Greenfoot
* Source files - WBC.zip

**Activity**

**Part I: Practice with Game Design - Outputs**

1. Download WBC.zip, create a new subfolder called WBC in your Greenfoot project folder, extract the files from the zip into your folder. Open the scenario (update the Java if needed). Make sure to compile if needed.
2. Explore the scenario, but make no changes to the code. Run it and see how it behaves. What do each of the classes represent? Do you understand how the source code works?

|  |
| --- |
| Insert Javadoc and regular comments describing what the code does as you watch the program function. As you go on reading, adjust your comments if you need to. |

1. Look at the source code for the WhiteCell class. It uses 3 methods - look them up in the API to see what they do. Notice that the setLocation() method uses the getX() and getY() methods to set parameters. When this happens, the execution direction of the code (or **code flow**) goes from the inside out. The rules for code flow in Java are:
   * Top to bottom
   * Right to left (when there is assignment involved)
   * Inside out (when there is nesting involved)

Using setLocation() rather than move() allows us to move in a direction we aren’t currently facing. Notice that checkKeyPress() is a **private** function. Remember that this means that another object in the world cannot invoke this method. This is because the method is not something that was intended to work outside of the class it is inside of - WhiteCell.

1. Look at the source code for the Bacteria class. You will notice this line:

getWorld().removeObject(this);

This is an example of method-chaining, calling one method on the return of another method. This line retrieves the world and the calls the removeObject() method on it. It also uses the keyword **this**, which allows an object to refer to itself / call a method on itself.

1. First let’s open the Lining class. We want the appearance of moving through the bloodstream without moving the white cell - so we will animate the lining. Here’s your first challenge:

|  |
| --- |
| * Make the Lining objects move continuously to the left by one cell (pixel) per act() cycle. Hint: look at the Bacteria class code for inspiration. * Make Lining objects disappear when they reach the left side of the screen. Hint: look at the Bacteria class code for inspiration. |

1. Open the Bloodstream class. Do the following:

|  |
| --- |
| * In the act() method, make new Lining objects appear on the top or bottom of the right side of the screen with a 1 percent probability. The world is 780 cells wide by 360 tall. |

1. Let’s add some enemies, a lose condition, and a win condition.

|  |
| --- |
| * Add a new class called Virus as a subclass of Actor. Set the image to virus.png. * In the Act() method Make viruses move left at 4 cells per act() cycle and rotate counter-clockwise. * In the Bloodstream class add a new method called spawnVirus() to add viruses at random spots on the right edge of the screen at a 1 percent probability. * In the WhiteCell class write a new method called checkCollision() that removes bacteria if you touch one and plays slurp.wav or your own sound that you have placed into the sounds folder of this scenario. * Modify checkCollision() to end the game if the white cell touches a virus. Have the scenario play game-over.wav if this happens. * Give the white cell the ability to go up and down. |

1. Let’s make the movement speed of the bacteria variable and create some red blood cells for ambience.

|  |
| --- |
| * In the Bacteria class declare a variable to control speed. Set it to a random integer from 1 to 3 when the bacteria are instantiated. In the act() method you will need to use the variable to adjust the speed of the bacteria. * Create a new class called RedCell and use the prepared image. Make the red cells just like bacteria in terms of variable movement but with a 6 percent spawn rate. * Give each red cell a random rotation when spawned. |

1. Let’s clean up the look of the game a bit.

|  |
| --- |
| * Create a new class called Border as a subclass of Actor and use the prepared image. * In the prepare() method of the Bloodstream class write code to create a border at (0,180) and another at (770,180). |

1. If you look at the code, you will see there are 2 problems. First the border partially obscures the starting position of the white cell. Next our object should appear below the border rather than on top of it. Let’s fix the problems:

|  |
| --- |
| * Find the code that spawns the white cell and alter its position. * Changing the position of the border requires the use of another method - find it using the API for the World class. Hint: we want to change the paint order. |

**Part II: Basics of Strings**

1. Let’s do one last thing: add a scoring system.

|  |
| --- |
| * Add a variable called score to the WhiteCell class. * Add 20 points to the score each time a bacteria is removed. |

1. We will use a method called showText() in the World class. Look it up in the API and you will see this signature:

**void** showText(String text, **int** x, **int** y)

**String** variables are special object types that:

* Store text in the form of characters, words, or sentences.
* Strings are surrounded by quotes and colored green in the Greenfoot IDE; we’ve used them before when playing sounds (“slurp.wav”)
* Each character in a String has an **index**, or position. Indexes start at zero, so the first letter is in position zero.
* Spaces count as characters.
* There are also special methods associated with String objects called **string operations**.

|  |
| --- |
| * Insert the line getWorld().showText(“Score: ”,80,25); so that it says Score: when you catch a bacterium. |

1. Now we need to add the score to the String. Adding two Strings or together to get a new String is an operation called **concatenation**. In Java we use the **+** operator between two Strings to concatenate them. Try this and see what happens:

We have a problem because the score contains an integer and not a String, so concatenation does not work. Changing one data type to another is called **casting** in Java and it can be rather complicated. More on casting later. Let’s convert the score to a String using this line:

|  |
| --- |
| * Integer.toString(score); //Casts int score to a String * Change the line getWorld().showText(“Score: ” + score,80,25); |

Actually, Java will automatically cast an integer to a String when you try to add the two. It assumes you are attempting to concatenate. This isn’t something Java does with many variable types, as we will see later on. So the code will work without the line to cast the integer to a String.

1. Add the following to the scoring rules:

|  |
| --- |
| * Lose 15 points if a bacteria escapes the screen. * Being hit by a virus doesn’t kill you - instead you lose 100 points. * We lose the game if our points fall below zero. In order to do this last one, we should move the score variable to our Bloodstream class. * We also need to initialize our score to zero when the world is built. * Add a method called addScore() to the Bloodstream class. Move the code to increment and display the score into this method. * Make sure to tweak the code in the other classes to get this all to work. * Display the score from the beginning of the game and not just when a bacterium is eaten. * Make sure you added Javadoc comments and basic comments throughout the code! |

1. Let’s add a timer that counts down. If we survive until the timer reaches zero, we win.

|  |
| --- |
| * Create the timer in two new private methods of the Bloodstream class called countTime() and showTime(). * If we survive 2000 cycles of the act() method, we should win the game. * Create a method called showEndMessage() and call it when the time runs out. Have it give you the final score for that game. |

**Conclusion**

Answer the following questions:

1. Codingbat exercises (testing your use of integer variables like score)
   1. <http://codingbat.com/prob/p154485>
   2. <http://codingbat.com/prob/p116624>
   3. <http://codingbat.com/prob/p184004>
2. I want to call the World method removeObjects() on an Actor object while inside the Actor class. How would I do this in one line using method chaining and dot notation?
3. Write the method calls for the following (use APIs if you need help:
   1. Inside an Actor object call the Greenfoot method getKey().
   2. Inside a World object call the isTouching() method on a particular object assigned to the variable lobster. Check if lobster is touching an object of class Crab.
4. What is the difference between a public method and a private method? When do I want to make a method private?
5. What is casting? When do we need to cast a variable?
6. What happens when the **+** operator is used between two integers? What about when it is used between two Strings? An integer and a String?