**CSCI145 Fall 2015 – Greenfoot Ch. 5 (30 points) Your name: Keila Calderon**

For **up to 30 extra credit points** added to your homework grade, please work through the exercises in Ch. 5 of the Greenfoot book. Note that this is an “all-or-nothing” type of assignment: in order for you to earn *any* extra credit, you must complete the *entire* assignment.

Be sure to go to your Greenfoot\_localRepo folder and do a **git pull origin master** before you begin any of the work below. Otherwise, record your responses in the spaces provided. In some cases, you may be asked to provide a screenshot (or two). ***Good luck – and have fun!!***

***IMPORTANT:*** At this point, I suggest you save this Word document in your Greenfoot\_localRepo folder, and periodically save, add, commit, and push. (That way, you’ll have it available on any computer you use, just like you’ll have your source code!)

**Greenfoot Exercise 5.1:** Open the scenario WBC-1 and run it. Describe what you observe.

|  |
| --- |
| In this scenario, there is a white blood cell present in a blood vessel with bacteria flowing towards the cell. The white blood cell is only able to move up and down and has no impact on the bacteria, nor do the bacteria impact the white blood cell. |

**Greenfoot Exercise 5.2:** For each of the classes in the scenario, write a short (one- or two-sentence) description about what they represent.

|  |
| --- |
| The class **Bloodstream** represents the “World” in which the actors WhiteCell, Bacteria and Lining live in. Bloodstream also makes use of the getRandomNumber to instantiate the bacteria flowing through.  The class **WhiteCell** represents an actor in the scenario which currently only carries the ability to move up or down 4 pixels with the arrow keys.  The class **Bacteria** represents another actor in the scenario which currently only carries the ability float along the bloodstream, rotating slowly.  The class **Lining** is an actor class in the scenario, which currently does nothing. |

**Greenfoot Exercise 5.3:** Open the source code for each of the four classes. Study the code and try to work out how it works. In the space below, note any section in the code that is unclear to you.

|  |
| --- |
| /\*\*  \* Float along the bloodstream, slowly rotating.  \*/  public void act()  {  setLocation(getX()-2, getY());  turn(1);    if (getX() == 0)  {  getWorld().removeObject(this);  }  } |

**Greenfoot Concept:** What is the purpose of the **setLocation** method?

|  |
| --- |
| The purpose of the setLocation method is to set an actor’s location to a position specified by x- and y-coordinates. |

**Greenfoot Concept:** What is the purpose of an **access modifier**?

|  |
| --- |
| The purpose of an access modifier is to determine who can see and call a method whether it is private or public. |

**Greenfoot Concept:** What is the purpose of a **private method**?

|  |
| --- |
| The purpose of a private method is to improve the structure of code by breaking tasks down into smaller subtasks. They are only visible within the class they are declared in therefore they are not intended to be called from outside. |

**Greenfoot Concept:** What is the purpose of the keyword **this**?

|  |
| --- |
| The purpose of the keyword **this** is to refer to the current object that is executing at the moment. |

**Greenfoot Concept:** What is the purpose of the **getWorld** method?

|  |
| --- |
| The purpose of the getWorld method is to give access to the world from an actor object. |

**Greenfoot Exercise 5.5:** Make the **Lining** objects move continuously left. They should move by one cell (one pixel) per act cycle. You can do this by copying the first line of the **Bacteria**’s **act** method, and changing the movement distance to 1. In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step.

**Greenfoot Exercise 5.6:** Make the **Lining** objects disappear when they reach the left edge. Again, an example of this is in the Bacteria class, and you can use this for guidance. In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step.

**Greenfoot Exercise 5.7:** Write a suitable **Javadoc** comment for the act method. You do not need @author and @version annotations in Javadoc comments for methods, but you should update the @author and @version annotations in the Javadoc comment for the Lining class.

**Greenfoot Exercise 5.8:** Make the **Lining** objects appear on the right side of the screen. Do this in the **Bloodstream**’s **act** method. They should appear with a one percent probability (on average, once every 100 act cycles). In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step.

Before you proceed, take a screenshot of your updated **Lining** class (at this point) and paste it in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use **auto-layout**).

|  |
| --- |
|  |

**Greenfoot Exercise 5.9:** Add a new class for a virus to your scenario (as a subclass of Actor). Call it **Virus**. There is an image already prepared in the scenario for you to use (see Figure 5.2). In the space below, paste a screenshot (or two) depicting the process of adding the **Virus** class to your WBC-1 scenario.

|  |
| --- |
| Step 1: Right Click on the Actor icon and click on New subclass  Step 2: Add the new class name and select the wanted image and click ok. |

**Greenfoot Exercise 5.10:** Extend the **Bloodstream**’s **act** method to add new virus objects at the right edge of the screen. Make the y-coordinate random. Make the probability of new objects 1 percent (one in a hundred act cycles). Test it. (Virus objects should occasionally appear at the right edge, but not yet move.) In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step.

Before you proceed, paste a screenshot of your updated **Bloodstream** class in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.11:** Make **Virus** objects move left and rotate (just like **Bacteria**). However, viruses move four cells per act cycle (not two), and they rotate counter-clockwise. In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step.

**Greenfoot Exercise 5.12:** Update the Javadoc comments for the **Virus** class and its **act** method.

Before you proceed, paste a screenshot of your updated **Virus** class in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.13:** Define a new ***private*** method in class **WhiteCell**, called **checkCollision**. The method body can initially be empty. Call this method from your **act** method.

**Greenfoot Exercise 5.14:** In the **checkCollision** method, add code that removes bacteria if the current WhiteCell object is touching any. (Use the **isTouching** and **removeTouching** methods, just like in the little-crab scenario.) In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step.

**Greenfoot Exercises 5.15 & 5.16:** Play a sound when removing bacteria. The “slurp.wav” sound is included again in the scenario—you can use this one for testing purposes, but try to find another sound effect WAV file on the internet; download this file and copy or move it to the sounds folder of your WBC-1 scenario folder. In your code, use a C-style or end-of-line comment to annotate the statement(s) you added/modified in this step. As part of this comment, please cite the Web address where you found the WAV file you used.

**Greenfoot Exercise 5.17:** Add similar code to the **checkCollision** method (another if-statement) to check whether we are touching a virus. If we are, play a sound (a sound called “game-over.wav” is included for his purpose), and then call the **stop** method of the Greenfoot class.

In the space below, offer an explanation (in your own words, and using complete sentences) of why it makes sense for the **checkCollision** method to be declared ***private***.

|  |
| --- |
| It is important that the checkCollision method be declared private because this way it can only be call from methods within the same class, making it invisible from the outside. If the method were to be declared public it would be open to other classes in the program making it accessible to be called on. |

Read the note at the bottom of p. 88, and investigate Chapter 9 (specifically Figure 9.2 and Section 9.5) as directed. In the space below, use your understanding of Figure 9.2 to briefly summarize why the game stops when the white blood cell is near a virus but is not actually touching it yet.

|  |
| --- |
|  |

Before you proceed, paste a screenshot of your updated **WhiteCell** class in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.18:** In your **Bacteria** class, add an instance variable of type **int**, named **speed**.

**Greenfoot Exercise 5.19:** In the Bacteria constructor, assign a random value to the speed variable, in the range from 1 to 3.

**Greenfoot Exercise 5.20:** Change the movement statement in your **act** method so that you subtract the variable **speed** from your x-coordinate (instead of substracting the value 2). Test, and use a C-style or end-of-line comment to annotate the statement(s) you added/modified in Exercises 5.19 and 5.20.

Before you proceed, paste a screenshot of your updated **Bacteria** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.21:** Add a new class called **RedCell**. You will find an image for it in your scenario. In the space provided below, paste a screenshot of the current World and Actor class diagrams at this point (or just paste a screenshot of the entire Greenfoot main window, as long as it includes both class diagrams.)

**Greenfoot Exercise 5.22:** Make the red cells move just like bacteria. That is, they move from right to left at variable speed, slowly rotating. There is one small difference: The speed range of red cells is only 1 to 2 (instead of 1 to 3). Our red cells are slow. Use a C-style or end-of-line comment to annotate the statement(s) you added/modified in Exercises 5.19 and 5.20.

**Greenfoot Exercise 5.23:** Extend your **act** method in class **Bloodstream** to create red blood cells. This is very similar to creating bacteria or viruses, but red blood cells are more frequent. Give them a *6 percent* chance of appearing.

**Greenfoot Exercise 5.24:** Initialize your red blood cells with a random rotation, using the statement shown on p. 90 in your book.

Before you proceed, paste a screenshot of your updated **Bloodstream** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

In addition, paste a screenshot of your **RedCell** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.26:** Add the code shown at the top of p. 91 to the **prepare** method of the **Bloodstream** class. Test it. What remaining problems do you observe?

|  |
| --- |
|  |

**Greenfoot Exercise 5.27:** In the **prepare** method of the **Bloodstream** class, find the place where the initial x-coordinate for the white blood cell is defined. Change it to a larger value, so that the cell is a bit further to the right.

**Greenfoot Exercise 5.28:** In the constructor for Bloodstream, add the following statement:

**setPaintOrder( Border.class );**

This will make the border objects appear “on top of” the other objects. Test it.  
  
Before you proceed, paste a screenshot of your updated **Bloodstream** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.29:** Add the ability for the white blood cell to move right and left as well as up and down.

**Greenfoot Exercise 5.30:** Add an instance variable called **score**, of type **int**, to your **WhiteCell** class.

**Greenfoot Exercise 5.31:** Add a statement to increment the score by 20 when we catch a bacterium.

**Greenfoot Exercise 5.32:** Look up the showText method from the World class. How many parameters does it have? What are they?

**Greenfoot Concept:** What is the purpose of the **String** data type? What is stored in variables of type **String**? Also: Is the String a *primitive* type, or is it a *reference* type? How do you know?

|  |
| --- |
| The purpose of the String data type is to represent text, such as words or sentences. Strings are written in double quotes. |

**Greenfoot Concept:** What is **string concatenation**, and what operator is used to concatenate two String objects together in Java? How does string concatenation differ from formatted printing using the **printf** method that you have learned in Ch. 2 of the Deitel book?

|  |
| --- |
| String concatenation merges two Strings into one. It is written with a plus symbol. |

**Greenfoot Exercise 5.33:** Insert the statement shown towards the middle of p. 93 so that it writes the word “Hello” to the screen when you catch a bacterium.

**Greenfoot Exercise 5.34:** Change the statement to write your name instead. Write this version of the statement in the space below.

|  |
| --- |
|  |

**Greenfoot Exercise 5.35:** Change the text again to show the word “Score:” (including the colon)

**Greenfoot Concept:** When String concatenation is used with a String followed by an integer (or perhaps an expression that evaluates to an integer), what happens to the integer? And what is the *data* *type* of the result of a String concatenated with an integer?

|  |
| --- |
| When String concatenation is used with a String followed by an integer, the integer value is converted to a String and then concatenated. |

**Greenfoot Exercise 5.36:** Show the score in your game.

Before you proceed, paste a screenshot of the current version of your **WhiteCell** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.37:** The game currently seems a bit too easy. Speed it up! Try the following:

* Make each Bacteria object move with a random speed (pixels per act cycle) between 1 and 5. (That is, when a given Bacteria object is instantiated, set its initial speed to be a random integer between 1 and 5.)
* Make the WhiteCell object move sideways with speed 4, and up and down with speed 8.
* Make each Virus object move with speed 8.
* Speed up the whole scenario using the speed slider at the bottom of the main window: set it to just above 50 percent.
* Experiment with the above speed values and settings to make the game challenging, but not impossible.

Before you proceed, paste a screenshot of the current version of your **Bacteria** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.38:** Move the score variable from WhiteCell to Bloodstream.

**Greenfoot Exercise 5.39:** Add a line to the Bloodstream’s constructor to initialize the score to zero. (This is not *strictly* necessary, because the default value for instance variables is zero, but it is still good practice.)

**Greenfoot Exercise 5.40:** Add a public method called addScore() to the Bloodstream class. Move your code that increments the score (and shows the score on screen) into this method. Note that you can now omit the call to the getWorld method – *in the space below, explain why (in your own words):*

|  |
| --- |
|  |

**Greenfoot Exercise 5.41:** Add a Javadoc comment to your new **addScore** method.

**Greenfoot Exercise 5.42:** In your **WhiteCell** class, add code to *call* the Bloodstream’s addScore method. Test it. Your scenario should now run again, and catching bacteria should score points.

**Greenfoot Concept:** What is the purpose of **parameterizing** methods? What does it actually *mean* to “parameterize” a method? As part of your answer, give at least one example from this section in the book.

|  |
| --- |
| The purpose of parameterizing methods is to add parameters which can make them more flexible and more useful. |

**Greenfoot Concept:** The title of Section 5.13 is “Abstraction: Generalizing the Scoring.” What do you think the word “**abstraction**” means, at least in the context of **parameterization**? (Hint: in this context, the word “abstract” essentially means the opposite of “specific” … or perhaps “specialized.”)

|  |
| --- |
| In the context of parameterization, I believe abstraction means that rather than programming a method to do one specific thing we can program it to do many similar things so that it can be used more generally in different situations. |

**Greenfoot Exercise 5.43:** Change your **addScore** method to expect a parameter for the points, as discussed above.

**Greenfoot Exercise 5.44:** Change the code in the **WhiteCell** class accordingly: pass a parameter to **addScore**, so that neutralizing a bacterium scores 20 points again.

**Greenfoot Exercise 5.45:** Add code in your **Bacteria** class so that you lose 15 points when a bacterium (that is, a single Bacteria object) exits the screen.

At the bottom of page 98, and also after reviewing Code 5.7, explain what the book means when they say “we can now use the **bloodstream** variable to call the **removeObject** method, instead of calling **getWorld()**”. You may need to go back and review p. 85 (section 5.3).

|  |
| --- |
|  |

**Greenfoot Exercise 5.46:** In **Bloodstream**, make a new **private** method called **showScore()**. It has *no parameters*, and it *returns nothing*. Move the **showText** statement that displays the score into this method. Where **showText** was called previously, call **showScore** instead now.

**Greenfoot Exercise 5.47:** In the constructor for **Bloodstream**, add a call to **showScore**. This will cause the score to be displayed at the start.

Before you proceed, paste a screenshot of the current version of your **Bloodstream** class source code in the space below. Make sure you have labeled all of your closing braces, and ensure all of your code is properly indented (use auto-layout).

|  |
| --- |
|  |

**Greenfoot Exercise 5.48:** Change your program so that touching a virus does not immediately end the game. Instead, the virus is removed and you lose 100 points. In the space below, explain why it is important to remove the virus.

|  |
| --- |
|  |

**Greenfoot Exercise 5.49:** Try using Greenfoot’s built-in sound recorder to make a new sound for touching a virus (see Chapter 10 in the Greenfoot book for more about sound recording in Greenfoot). Be creative!

**Greenfoot Exercise 5.50:** Move the game-over functionality (that is, the code responsible for playing the game-over sound and stopping Greenfoot) into the **addScore** method, so that the game is over if the score falls below zero. (You will need an *if-*statement after changing and displaying the score.)

**Greenfoot Exercise 5.51:** In class **Bloodstream**, add an instance variable of type **int**, called **time**.

**Greenfoot Exercise 5.52:** In the constructor for **Bloodstream**, initialize the **time** to 2000. (The idea is that when you start the game, your objective is to survive for 2000 act cycles.)

**Greenfoot Exercise 5.53:** Define a new **private** method called **showTime** that displays the time left (that is, the value of the **time** variable) near the top right of the screen. Call this method from the constructor to show the initial time.

**Greenfoot Exercise 5.54:** Define a new **private** method called **countTime** that decrements the **time** by 1 every time it is called, and then shows the current time (by calling **showTime**). Call this method from your **act** method.

**Greenfoot Exercise 5.55:** In your **countTime** method, add an *if*-statement that stops execution when the timer reaches 0.

**Greenfoot Exercise 5.56:** Add a new ***private*** method called **showEndMessage**. When called, it displays a message telling us that we won, along with how many points we have, near the middle of the screen. But display something a bit different from the book’s suggestion of “Time’s Up – You win!” (Because that sounds weird, doesn’t it? Last I checked, when your time is up, you typically *lose*, right?)

Call this method when the timer runs out (and thus the player wins because they survived the entire time).

At this point, use Git Bash to **add, commit, and push** your *Greenfoot\_localRepo* repository to your Bitbucket account. Don’t forget to navigate to the root folder of your local repo before you add, commit, and push. Use the following commands:

**git add –A**

**git commit –m "completed up to exercise 5.56"**

**git push origin master**

***Continued on next page***

Before you proceed, save your scenario under the name **WBC-Ch5ex57**. Once you’ve done that, then do exercise 5.57 as follows (and make sure you are using the scenario named **WBC-Ch5ex57**).

**Greenfoot Exercise 5.57:** Experiment with the parameters of this game to make it more playable and interesting. You can adjust things like:

* The speed at which each actor moves
* The number of points you gain and lose
* The amount of time available
* The sound effects
* The frequency of new actors appearing
* The execution speed of the scenario
* And anything else you can think of!

After you’re finished with exercise 5.57, please use Git Bash to **add, commit, and push** your *Greenfoot\_localRepo* repository to your Bitbucket account. Don’t forget to navigate to the root folder of your local repo before you add, commit, and push. Use the following commands:

**git add –A**

**git commit –m "completed up to exercise 5.57"**

**git push origin master**

Before you proceed, save your scenario under the name **WBC-Ch5ex58**. Once you’ve done that, then do exercise 5.57 as follows (and make sure you are using the scenario named **WBC-Ch5ex58**).

**Greenfoot Exercise 5.58:** Change the scenario’s image to put the game into an entirely new setting. For example, you could make it a spaceship flying through space, collecting astronauts and avoiding asteroids, or a rabbit running over a field while catching carrots and avoiding dogs. You can do anything you like here – make up something and BE CREATIVE.

If you wish, you can try working with the assets that have been posted to Blackboard (**look under Course Documents > Final Project Info > Sprite sheets and other game assets**).

If you are happy with your idea and images, it would be good to change the class names, too. Moreover, different settings might give you different ideas for adding functionality to your game.

After you’re finished with exercise 5.58, please use Git Bash to **add, commit, and push** your *Greenfoot\_localRepo* repository to your Bitbucket account. Don’t forget to navigate to the root folder of your local repo before you add, commit, and push. Use the following commands:

**git add –A**

**git commit –m "completed up to exercise 5.58"**

**git push origin master**

**Submission instructions:**   
In addition to pushing your work to Bitbucket as described above, please **save** this file (**renaming** it to replace *FirstnameLastname* with your own name) and **upload** this to the assignment page on Blackboard before the deadline (Monday, 2 Nov 2015 at 11:59pm).