# Introduction

## Project Introduction

## Programming Language

Java is a generic term for the Java object-oriented programming language and the Java platform that was introduced by Sun Microsystems in May 1995. It was co-developed by James Gosling and colleagues, and was officially launched in 1995.

## Revision Control

Git is an excellent distribution version control system. The version control system can keep a history of a collection of files and can roll back the file collection to another state (history state). Another state can be a different file, it can be a different file content. In a distributed version control system, everyone has a complete source code (including all the history of the source code information) and can operate on the local data. The distributed version control system does not require a centralized code repository.

## 1.4 Debugger

## 1.5 Build Tools

## 1.6 Test Tools

System requirements

To follow along and try out the code for this tutorial, you need a working installation of Sun's JDK 1.5.0\_09 (or later) or the IBM developer kit for Java technology 1.5.0 SR3. For the sections on running JUnit 4 in Eclipse, you need a working installation of Eclipse 3.2 or later. For the section on Ant, you need version 1.6 or greater.

The recommended system configuration for this tutorial is as follows:

A system supporting either the Sun JDK 1.5.0\_09 (or later) or the IBM developer kit for Java technology 1.5.0 SR3 with at least 500MB of main memory

At least 20MB of disk space to install the software components and examples covered

The instructions in the tutorial are based on a Microsoft Windows operating system. All the tools covered in the tutorial also work on Linux and UNIX systems.

（<https://www.ibm.com/developerworks/java/tutorials/j-junit4/index.html>）

# Relative Technology

## 2.1 Third-Party Technology

Where to get, and how to build and install, any third-party packages needed by your code (for packages that are not already on the Physics Computational Lab machines).

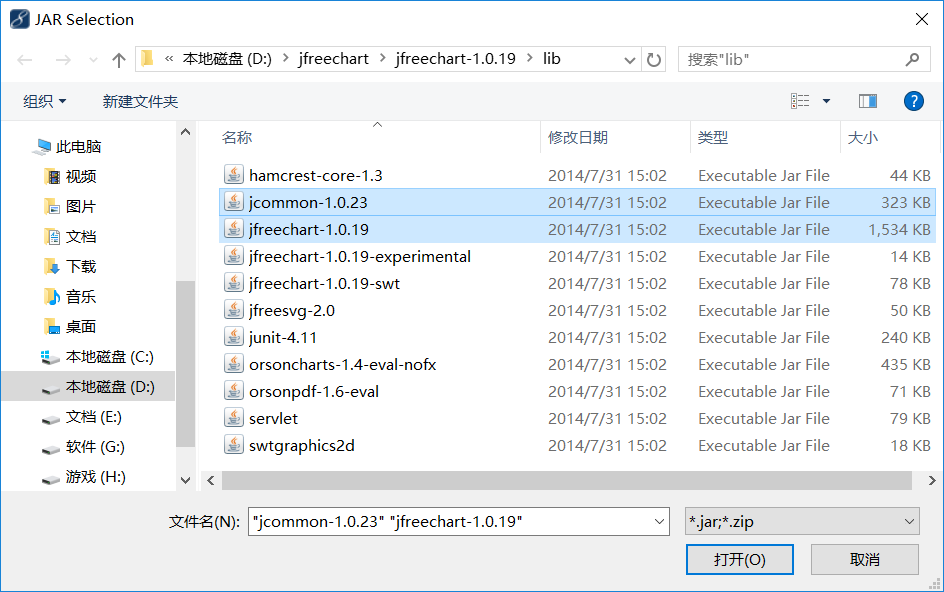
We use Jfreechart to draw the line chart of average density of the result.

Before we use it, we need to install it into our project. The procedure is as followed:

1. Download the latest version of Jfreechart by the website : <https://sourceforge.net/projects/jfreechart/files/1.%20JFreeChart/1.0.19/>. Then unzip it.

2. Next step is to import the necessary jar files into the project. If we use eclipse, we can select the current program, right click it, select “build path- configure build path”.

3. In “libraries”, click “add external jars”, find the path of jfreechart and then go into the lib, select “jcommon-1.0.23” and “jfreechart-1.0.19”. Add them into reference libraries.



# Build the code

## 3.1 Build for main program

1. To make it a runnable file by command line from eclipse, we can find “export” button from file menu.
2. We will come into a new window by clicking export. Then click “java” to go to a submenu.
3. Now we can see “JAR file” option. We need to pick that option and click “next” button at the bottom of the window.
4. Choose a proper destination for your JAR file by clicking browse and the click “next”.
5. The last step is to click “finish”.

## 3.2 Build for Junit test.

1. Right click on the “TestProject.java” Junit class.
2. Find “Run as” option.
3. Run the test by clicking JUnit Test.

# Run the code

1. Fisrtly, we need to build the project into a runnable jar file with the help of eclipse or other IDE.
2. Next, copy the runnable jar to the dictionary we want, where java environment has been installed.
3. Last, use the command “java -jar xxxx.jar” to run the program successfully.

# Test the code

## 5.1 Test Plan

Test Case

|  |  |  |  |
| --- | --- | --- | --- |
| Items to be Tested | Features to be Tested | Feature Type | Level of Risk |
| WritePPM() | Write color array to a ppm file | Functionality | Low |
| WriteText() | Write the density array to a txt file | Functionality | Low |

## 5.2 Test Case

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case Name** | **TesOutputPPM** | | **Case No.** | **005** |
| **Case Version** | **V1.0** | | | |
| **Pre Condition** | **No** | | | |
| **Post Condition** | **After testing 005, please go to test 006** | | | |
| **Test Point** | **Testing Procedure** | **Expected Result** | | **Testing Result** |
| **Test WriteText() method** | 1. **Create a 10 by 10 2-dimensional matrix.** 2. **Write this matrix using WriteText() method to a txt file.** 3. **Read average density to variable “a1”.** 4. **Read the data from this txt file back to a 10 by 10 array called “text”.** 5. **Compare an defined double value with “a1”.** 6. **Compare this matrix and array “text” to see if they are equal using assertArrayEquals().** | **A1 equals to the value we defined. And the matrix we defined is totally equals to the array we read from txt file.** | | **A1 equals to the value we defined and the matrix equals to array “text”.** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Case Name** | **TesOutputText** | | **Case No.** | **006** |
| **Case Version** | **V1.0** | | | |
| **Pre Condition** | **No** | | | |
| **Post Condition** | **No** | | | |
| **Test Point** | **Testing Procedure** | **Expected Result** | | **Testing Result** |
| **Test WritePPM() method** | 1. **Create a 3 by 3 2-dimensional array called “matrixInit”.** 2. **Write this array using WritePPM() method to a ppm file.** 3. **Read this ppm file back to a 3 by 9 array called “text”.** 4. **Compare this array “text” and another array we defined before called “matrixTest” to see if they are equal using assertArrayEquals().** | **“matrixTest” we defined is totally equals to array “text” we read from ppm file.** | | **“matrixTest” equals to array “text”.** |

# Summary of key design decisions and reasons

## 6.1 Input

The input part of the code is designed to read data from a dat file which contains the height, the width(these two are in the first line) and the data(2D array, the other part of the file).

The class used to get those data is BufferReader. It has two ways to read data: first choice is read character by character while the second choice is read line by line. It is clearly to see that read line by line run quicker than former one. The data we get from a line is in String type and we have to separate this String into the data format we need and make it easy approach by main function.

We store the height and width into an array. The details is: we change data type from String to String array to int array. This is just in one line, so we read once and get result.

We store the data(the content of landscape) into an 2D array. To make this works, we change data type from String to String array to int array to 2D int array. As there are lines of data, we use a condition test to check the data that we read from is available( there still have something to be read), and when we finish read the whole file, we get the data and return to main.

## 6.2 Algorithm Section

2.1 initialize the animal information

First two animal classes (hare and puma) have been declared to store the information of these.

In the 2 classes, they contain the basic static statistics information (like birthrate, predationRate and so on), the location on the map and the density value.

And then, with the help of the map architecture(where contains land and water), we give all the land nodes of pumas and hares some random values which are between 0 and 5 so that all the land nodes have the initial values for algorithm.

2.2 make the model of algorithm

Next step is to achieve the algorithm. In order to get the right model of puma and hare, we need to calculate every node's neighbor first because accroding to the formula [picture of it], [前四邻居之和] is a necessary part and the number of neighbor is another essential part.

The core part of the algorithm is how to get the correct number of neighbor

Our method is to calculate different situations of the location for the map

(1)

When we find the current node is one of the 4 vertex nodes (in the red circles), the neighbour number of current node is 2, and then detect how many water nodes the neighbours contain. If there is no water node, the final neighbour number is 2. If one water node, the neighbour number is 1. If two water node, the number is 0. After that we the calculate the sum of the density of the neighbours for the current node.



(2)

When we find the current node is on the boundary but not the vertex ones (in the ovals), the neighbour number is 3, and then dealing with the water node just like part(1) do. Last, we calculate the sum of the density of the neighbour nodes.



When we find the current node is in the middle part of the map which means it is not on the border lines (in the shadow part), the number of neighbours is 4. Next, the water node should be considered and the calculation will be done finally.



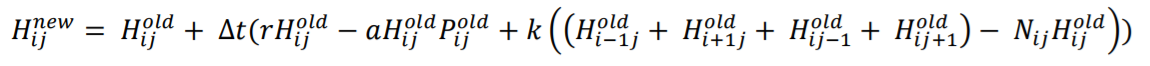
2.3 calculate the next generation of puma and hare

After we get the number of the neighbors and the value respectively, we have the result of the next formula:





We can calculate the result of density of next generation using the formula (NO.)as followed.





In the 2 formulas, we can use the value of parameters in table(No.):

|  |  |
| --- | --- |
| parameters | values |
| r: the birth rate of hares | 0.08 |
| a: the predation rate at which pumas eat hares | 0.04 |
| b: the birth rate of pumas per one hare eaten | 0.02 |
| m: the puma mortality rate | 0.06 |
| k: the diffusion rates for hares | 0.2 |
| l: the diffusion rates for pumas | 0.2 |
| Δt: the size of the time step | 0.4 |

## 6.3 Output

The class WriteToPPM has to output the density array of hares and pumas to ppm files every T cycle. The first step is to read the result array which is given by calculation function for the hares and pumas. Then we need to a convert the amounts of the two creature to a certain color, and save the RGB text information to a ppm file which can be displayed by a certain software colorful. After we receive a 2-dimension array, every element in the result array is specified to a certain color. Then we create a new color array for each pixel which store its RGB value.

In ppm file, we need to write the information at begging of the file for other software to display the ppm file to a colorful picture properly. The program write “P3” at fist line to indicate the file encoding type. Then we write the size of matrix wrote in this file. And we need to put the range of RGB value to the next line. After all above, we can write the color matrix to this file.

In WriteToPPM() method, there are 3 parameters. The first one “resultArray” is the density results of either hare or puma in a 2-dimension array. “Animal” is to specify which animal density we are writing. Hare uses pink color, puma uses blue. T is the time interval that it needs to output a ppm file.

We also need to record the density of two creatures every T generations. So that we have another class called WriteToText() which writes the density array and average density value at this generation without converting to a color array to a txt file. There are four parameters. “resultArray” is the same with previous method to record the density. “average” is the average density of one animal at present. “animal” is to indicate which animal we are recording. “T” tells us the time interval that we output the txt file. And we make the average density of every T generations to a line chart which uses a third-party library called jfreechat and jcommon.

# Reference

# Appendix