AG2411 – GIS Algorithms
Takeshi Shirabe
Geoinformatics, KTH
2021-11-08

Exercise 1
Implementing routines to read and write raster data files

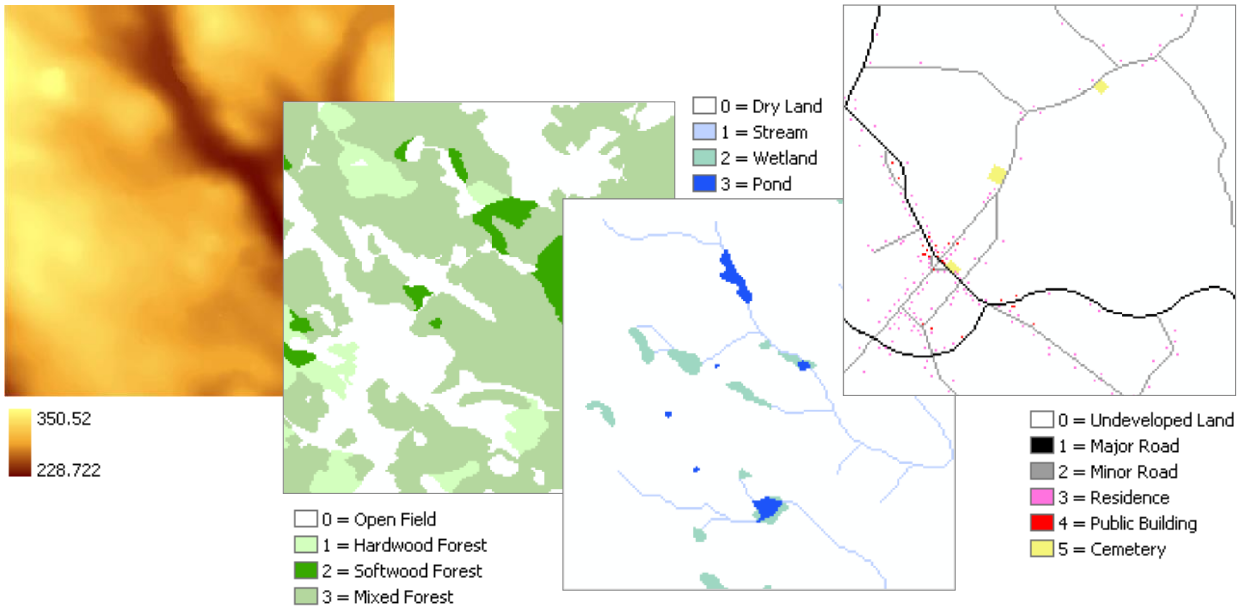
In the first three exercises of this course, you will build a raster-based geographic information system (GIS) from scratch, that is, without using any existing GIS software or any module of it. It is a very primitive GIS, as its functionality is limited to reading, writing, visualizing raster data and performing several map algebraic operations. Through these exercises, however, you will learn how a GIS works at a lower level as well as how to implement basic raster GIS algorithms.

We have chosen Java as the programming language of this course for two reasons. First, Java is one of the simplest object-oriented programming languages. Object orientation is a paradigm adopted by many modern programming languages. The other reason is a more practical one: the teaching staff can help you learn programming in Java. However, if you do not feel comfortable with this choice, you may use another general-purpose language such as C++. **But before doing so, please contact the instructor.**

You will begin with a simple exercise to familiarize yourself with programming in a (raster) GIS context. More specifically, you will write a Java program 1) to read a raster data file in ASCII format exported from ArcGIS and store the raster layer in a computer memory and 2) to save a raster layer in an ASCII file importable to ArcGIS.

PREPARATION

Create a working directory in which you will save all data and code related to this exercise. Download from CANVAS a set of raster layers representing elevation, vegetation, hydrology, and development of a small study area called "Brown's Pond," located in the village of Petersham, Massachusetts, USA. Use ArcGIS to see all the layers and then covert them into ASCII files and save them in your workspace.



TASKS

Note that there is no self-study guide, step-by-step instructions, or sample code for this exercise. Come to the lab and use all kinds of learning resources including your teaching staff, reference books, and past handouts to get ideas on how to approach to this assignment.

Do the following tasks.

1. Create a new project and name it "AG2411_Raster".

2. In this project, create a new package and name it "kth.ag2411.mapalgebra".

3. In this package, create a class called **Layer** from which all raster layers are instantiated. It should be saved as "Layer.java" (case sensitive!).

   This class should have the following attributes and methods.

Layer
+ **name** String + **nRows** int + **nCols** int + **origin[]** double[] + **resolution** double + **values** double[] + **nullValue** double
+ **Layer(String, String)** + **print(): void** + **save(String): void**

<Attributes>
- The **name** attribute stores a String object representing the name of a layer to be instantiated from this class.
- The **nRows** attribute stores an int value representing the number of rows of the layer.
- The **nCols** attribute stores an int value representing the number of columns of the layer.
- The **origin** stores an array of two double values representing the coordinates of the lower left corner of the layer.
- The **resolution** attribute stores a double value representing the resolution of the layer.
- The **values** attribute stores an array of double values that constitutes the layer. Alternatively, this attribute stores a two-dimensional array of double values that constitutes the layer. The choice is up to you.
- The **nullValue** attribute stores a double value that is designated as no value. ArcGIS uses -9999 by default.

<Methods>
- The **Layer** constructor method takes two String objects, one representing the name of a layer and the other representing the name of an ArcGIS ASCII GRID file, checks if the file exists and is correctly formatted, instantiates a new **Layer** object, and assign values (or objects) to all its attributes.
- The **print** method takes no parameter, prints all the values of the Layer object on the console, and returns nothing.
- The **save** method takes a String object representing an output file name, creates a new file with that name, saves the Layer object in it, and returns nothing.

Your code may look like…

```
public class Layer {

	// Attributes (This is complete)
	public String name; // name of this layer
	public int nRows; // number of rows
	public int nCols; // number of columns
	public double[] origin = new double[2]; // x,y-coordinates of lower-left corner
	public double resolution; // cell size
	public double[] values; // data. Alternatively, public double[][] values;
	public double nullValue; // value designated as "No data"

	//Constructor (This is not complete)
	public Layer(String layerName, String fileName) {

		// You may want to do some work before reading a file.

		try {  // Exception may be thrown while reading (and writing) a file.
			// Get access to the lines of Strings stored in the file

			// Read first line, which starts with "ncols"

			// Read second line, which starts with "nrows"

			// Read third line, which starts with "xllcorner"

			// Read forth line, which starts with "yllcorner"

			// Read fifth line, which starts with "cellsize"

			// Read sixth line, which starts with "NODATA_value"

			// Read each of the remaining lines, which represents a row of raster
			// values

		} catch (Exception e) {
			e.printStackTrace();
		}

	}

	// Print (This is complete)
	public void print(){

		//Print this layer to console
		System.out.println("ncols         "+nCols);
		System.out.println("nrows         "+nRows);
		System.out.println("xllcorner     "+origin[0]);
		System.out.println("yllcorner     "+origin[1]);
		System.out.println("cellsize      "+resolution);
		System.out.println("NODATA_value  " + nullValue);

		for (int i = 0; i < nRows; i++) {
			for (int j = 0; j < nCols; j++) {
				System.out.print(values[i*nCols+j]+" ");
			}
			System.out.println();
		}
	}

	// Save (This is not complete)
	public void save(String outputFileName) {
		// save this layer as an ASCII file that can be imported to ArcGIS
	}
}
```

4. Compile the **Layer** class.

5. Create a class called **Ex01** in the kth.ag2411.mapalgebra package, copy the following code to it, and save it as "Ex01.java".

```
public class Ex01 {
	public static void main(String[] args) {
		if(args.length == 3){
			//Instantiate a layer
			Layer layer = new Layer(args[0], args[1]);
			//Printing it on the console
			layer.print();
			//Saving it to the file output.txt
			layer.save(args[2]);
		}
		else {
			System.out.println("Too many or few arguments......");
		}
	}
}
```

   This class has nothing but a **main** method (just like "HelloWorld" class), which takes an array of three arguments representing 1) the name of an input layer, 2) the name of an existing file that stores this input layer, and 3) the name of a new file that will store this layer. Then, it will print the all values of the input layer and makes a copy of the input file with a different name.

6. Compile and test the **Ex01** class.

SUBMISSION

Submit the following.
- Source code of **Layer** class (i.e., Layer.java) and of **Ex01** class (i.e., Ex01.java).

DUE DATE

To avoid penalty points, your submission must be made through **CANVAS** by **November 14, 2021, 11:59 PM**.

HELPS

1. Importing classes

In the first two laboratory sessions or elsewhere, you have learned that a **class** is an **abstract data type** from which **objects** are **instantiated**, and that there are a lot of built-in classes in Java. Some classes are very versatile and thus the developer of the Java language (SunMicrosystems and Oracle) has designed them to be immediately accessible to all programmers. Others have more specialized functionality and thus are stored in special packages, from which they need to be explicitly "imported" before being used (usually at the very beginning of the code).

Indeed, "import" is the keyword for importing a class (or a set of classes) from the package storing it. For instance, the Scanner class used in HelloWorld3.java is located in the package **java.util**. This was why the code began with:

```
import java.util.Scanner;
```

You can import classes that have been developed by other programmers, in the same manner. For example, the following statement imports the Point class stored in the *kth.ag2411.intro* project (see Exercise 00).

```
import kth.ag2411.intro.Point;
```

Layer.java may need the following classes.

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
```

2. Reading from a file

The constructor of the Layer class needs to read a file. This can be facilitated by the **File**, **FileReader**, and **BufferedReader** classes as follows.

```
// This object represents an input file, elevation.txt, located at ./data/.
File rFile = new File("./data/elevation.txt");

// This object represents a stream of characters read from the file.
FileReader fReader = new FileReader(rFile);

// This object represents lines of Strings created from the stream of characters.
BufferedReader bReader = new BufferedReader(fReader);

// Read each line of Strings
String text;               // stores each line of Strings temporarily
text = bReader.readLine(); // first line is read
System.out.println("this is the 1st line: " + text);
text = bReader.readLine(); // second line is read
System.out.println("this is the 2nd line: " + text);

// Read until the last line when the number of lines is not known
String text = bReader.readLine(); // first line is read
int count = 1;
while (text != null) { // Repeat the following until no more line to be read
       System.out.println("this is the " + count + "th line: " + text);
       text = bReader.readLine(); // next line is read
       count = count + 1; // keep track of the number of lines so far
}
```

3. Writing to a file

The save method of the Layer class needs to write to a file. This can be facilitated by the **FileWriter** class as follows.

```
// This object represents an output file, out.txt, located at ./data/.
File file = new File("./data/out.txt");

// This object represents ASCII data (to be) stored in the file
FileWriter fWriter = new FileWriter(file);

// Write to the file
fWriter.write("ncols         "+nCols+"\n"); // "\n" represents a new line
```

4. Methods of String

The String class has a number of useful methods (https://docs.oracle.com/javase/9/docs/api/java/lang/String.html). We may use some of the following in developing our Layer class.

- `String substring(int beginIndex)`
  Returns a new string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.
  e.g.
  ```
  String aLine = "ncols         4";
  String aLine_less_ncols = aLine.substring(5); // "         4"
  ```
- `String trim()`
  Returns a copy of the string, with leading and trailing whitespace omitted.
  e.g.
  ```
  String aLine = "ncols         4";
  String aLine_less_ncols_less_space = aLine.substring(5).trim(); // 4
  ```
- `String[] split(String delimiter)`
  Divides this string into tokens (≈ words) based on the specified delimiter (e.g., white space), and return an array of the tokens.
  e.g.
  ```
  String aLine = "10 20 10 10";
  String[] values = aLine.split(" "); // {10, 20, 10, 10}
  ```

Note: you may need to convert a String object representing a number to a value of the corresponding primitive data type (e.g. double). This can be done using a special method that is not provided by the String class. More details will be given during the lab.

5. Handling exceptions

It sometimes happens that you try to get access to a file that does not exist and your program terminates with a lengthy error message. It will be embarrassing if this actually happens not to you but to a user of your program.

In Java, some problematic event that a method encounters (but cannot handle) during its run time is called an **exception**. If this happens, the method will throw a signal, which is also referred to as an exception.

In order to respond to or handle an exception without letting the program hang up, Java programmers use a **try**…**catch** block, as shown below.

```
try {
        // do something
} catch (Exception e) {
        // do something
}
```

The try block encloses lines of code that may throw an exception. If an exception is thrown from the try block, the catch block will receive it and execute the line of code that encloses. For example, the catch block may include the following statement, which, if an exception is thrown, will print where it occurred and what kind of exception it was.

```
e.printStackTrace();
```

This is useful for **debugging**—fixing errors in code—which may be the most time-consuming and frustrating task in programming.

REFERENCES

Scanner class documentation
https://docs.oracle.com/javase/8/docs/api/java/util/Scanner.html
File class documentation
https://docs.oracle.com/javase/7/docs/api/java/io/File.html
FileReader class documentation
https://docs.oracle.com/javase/8/docs/api/?java/io/FileReader.html
BufferedReader class documentation
https://docs.oracle.com/javase/8/docs/api/java/io/BufferedReader.html
FileWriter class documentation
https://docs.oracle.com/javase/8/docs/api/index.html?java/io/FileWriter.html
Exception Handling
http://download.oracle.com/javase/tutorial/essential/exceptions/index.html