### Java File I/O (Reading and Writing)

<https://www.youtube.com/watch?v=hgF21imQ_Is&list=PLkP0ieAtO8LkmQN_4PnxKvDvTaGEEEIhn&index=28>

#### Writes the file

import java.io.\*; //need to access reading and writing files

try { //writing a file

BufferedWriter bw = new BufferedWriter(new FileWriter("C:\\Users\\kotag\\OneDrive - Wayne State College\\EclipseWorkSpace\\RWTest\\output.txt")); //path to write the file, even if the file doesn’t exist it will write the file in the location, if the file is already made every time you run this it will rewrite the file

bw.write("Java\n");

bw.write("Is\n");

bw.write("Cool!\n");

bw.close(); //needs to close to avoid future mistakes

} catch (IOException e) {

return;

}

#### Reads the file

try { //reading a file

BufferedReader br = new BufferedReader(new FileReader("C:\\Users\\kotag\\OneDrive - Wayne State College\\EclipseWorkSpace\\RWTest\\output.txt")); //path of the file we want to read

String s;

while((s = br.readLine()) != null) { //while each line read in the file isn’t null print out s

System.out.println(s);

}

br.close();

} catch(IOException e) {

return;

}

#### Does both

try { //copying a file

BufferedWriter bw = new BufferedWriter(new FileWriter("C:\\Users\\kotag\\OneDrive - Wayne State College\\EclipseWorkSpace\\RWTest\\output-copy.txt"));

BufferedReader br = new BufferedReader(new FileReader("C:\\Users\\kotag\\OneDrive - Wayne State College\\EclipseWorkSpace\\RWTest\\output.txt"));

String s;

while((s = br.readLine()) != null) {

bw.write(s + "\n");

}

br.close();

bw.close();

} catch(IOException e) {

return;

}

### Java Generics

<https://www.youtube.com/watch?v=h7piyWnQbZA&list=PLuVT2Ug8ISOUeumoUczDqraT_EO6qFdWt&index=8>

<https://www.geeksforgeeks.org/generics-in-java/#:~:text=Generics%20means%20parameterized%20types.,work%20with%20different%20data%20types>.

* Generics - parameterized types. The idea is to allow types (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types
  + We are creating one class to take in different variables
  + The class being passed in needs to be an object type, so no primitive types

public class Main {

public static void main(String[] args) {

MyClass<Integer> object = new MyClass(10); //<Integer> specifies that we are passing in an Integer

MyClass<Double> object2 = new MyClass(20.0); //<Double> specifies that we are passing in an Double

object.ShowType();

object2.ShowType();

}

}

public class MyClass<T> { //<T> placeholder for the class being passed in

T object;

public MyClass(T object){ //Constructor

this.object = object;

}

public void ShowType() {

System.out.println(object.getClass().getName());

}

}

#### Multiple Generics

public class Main {

public static void main(String[] args) {

MyClass<Integer, Double> object = new MyClass(10, 20.0); //it now requires to pass in 2 variable types

object.ShowType();

}

}

public class MyClass<T, V> { //add V for another variable type, You can have as many of these as you want as long as they aren’t keywords or already in use

private T object;

private V object2;

public MyClass(T object, V object2){

this.object = object;

this.object2 = object2;

}

public void ShowType() {

System.out.println(object.getClass().getName());

System.out.println(object2.getClass().getName());

}

}

#### Specific Generics

public class Main {

public static void main(String[] args) {

NumericFns<Integer> iObject = new NumericFns<>(4); //can only use Number Object class, so String wouldn’t work

System.out.println(iObject.square());

}

}

public class NumericFns<T extends Number> { //bounds our generic to Number which has Integer, Double, and Float

T object;

public NumericFns(T object) {

this.object = object;

}

public double square() {

return object.intValue() \* object.doubleValue(); //\* can only be a applied to numeric values

}

}

public class Main {

public static void main(String[] args) {

NumericFns<Integer> iObject = new NumericFns<Integer>(6);

NumericFns<Double> dObject = new NumericFns<Double>(-6.0);

System.out.println(iObject.absEqual(dObject));

}

}

public class NumericFns<T extends Number> {

T num; //for this class T will always be Integer because iObject was called first

public NumericFns(T object) {

this.num = object;

}

boolean absEqual(NumericFns<?> object) { //? Means we can pass in any Number bounded class

if(Math.abs(num.doubleValue()) == Math.abs(object.num.doubleValue())) {

return true;

}

return false;

}

}

### 6 Must Know Java Array Methods

<https://www.youtube.com/watch?v=86B96Fy6j6U&list=PLuVT2Ug8ISOUeumoUczDqraT_EO6qFdWt&index=2>

<https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html>

#### asList

String [] arr = {"chicken", "bacon", "ham"}; //String Array

List<String> list = Arrays.asList(arr); //takes the array and turns it into a list

for(String s : list) { //for every s in list

System.out.println(s + " ");

#### fill

int[] arr = new int[5]; //has five elements, each element is set to a default value of 0 or null

Arrays.fill(arr, 100); //sets all of the elements to 100

#### copyOf

int[] arr = new int[5];

int[] arr2 = Arrays.copyOf(arr, arr.length); //copying Array arr and the number of elements in the array into arr2

#### Equals

Arrays.equals(arr, arr2) //returns a boolean if it is true

#### sort

int[] arr = {7, 2, 12, 32, 26};

Arrays.sort(arr); //sorts through the array from lowest value to highest value. The sorting method works best with primitive types

#### binarySearch

int[] arr = {8, 10, 12, 14, 19, 27}; //has to be a sorted array

System.out.println(Arrays.binarySearch(arr, 19)); //prints out the index of where the element is, if it can’t find the element it will return a -1, if the array had a repeat of a value this would return the index of one of those values