# The End is Near!



#### **Course Evaluations in Miverva**

# The static and final keywords

# Static

# We use the static keyword when we want a variable or method to be accesible without creating an instance of a class

• A static variable will have the same value for all of the instances of a class

```
public class Node{
 1
 2
           // non static variables are unique to each instance of this class
 3
           private String data;
 4
           private Node next;
 5
 б
           // static variables are the same for every instance of this class
 7
           private static numberOfNodes = 0;
 8
 9
           public Node(String input_data){
10
               this.data = input_data;
               this.next = null;
11
12
               // we can access static variables inside non static methods
13
               numberOfnodes += 1;
14
15
           }
16
       }
17
```

- Only one copy of the static variable exists in the computer's memory.
- If you change the value of a static variable, this change will be seen in **all** instances of the class.

#### Static

# We use the static keyword when we want a variable or method to be accesible without creating an instance of a class

• A static method can be called without creating an instance of a class

```
public class Node{
 1
 2
           // non static variables are unique to each instance of this class
           public String data;
 3
 4
           public Node next;
 5
 6
           // static variables are the same for every instance of this class
 7
           public static numberOfNodes = 0;
 8
           public Node(String input_data){
 9
10
               this.data = input_data;
11
               this.next = null;
12
13
               // we can access static variables inside non static methods
               numberOfnodes += 1;
14
15
16
           // static methods of this class can be called as Node.static_method_name
17
           public static int NumberOfNodesSoFar(){
18
19
               return numberOfNodes;
20
21
22
```

- Inside the class file where they are declared, you can call static methods directly by their name
- Outside that file, you should call them as ClassName.staticMethodName(arguments)
- Only one copy of the static method exists in the computer's memory.

# Static

#### Some rules for using static methods and variables

- A non-static method can access: static methods and variables
- A non-static method can access: non-static methods and variables
- A static method can access: static methods and variables
- A static method **cannot** access: non-static methods and variables. It doesn't have access to the this reference.

# **Static + Final**

#### The final keyword is used when we want the value of a static variable to be constant

```
1 // The values of the following variables will remain unchanged over the execution of the program
2 static final double PI = 3.141592653589793;
3 
4 static final double PLANCK_CONSTANT = 6.62606896e-34;
5
```

This is usually useful when you want to precompute certain numbers and keep them constants during the execution of your program.

```
1 // The values of the following variables will remain unchanged over the execution of the program
2 static final double PI = precomputePi();
3
4 public static double precomputePi(){
5 // execute some algorithm for computing pi
6 }
7
```

#### How a Hashtable works

#### Think of a Hashtable as a dictionary

Key (String)	Value (String)
"brillig"	"the time when you begin broiling things for dinner"
"slithy"	"blend of slimy and lithe"
"gyre"	"to whirl"
"gimble"	"to make a face"
"burble"	"to make vocalized bubbles with the mouth"
"chortle"	"to chuckle and snort at the same time"
"mimsy"	"flimsy and miserable"

We use a Hastable to associate a key to a value

Keys, as well as values, can be of any type

Key (String)	Value (Long)
"Yogi"	869 567 3212
"Asako"	365 092 3744
"Marcel"	438 756 3321
"Ante"	438 887 1414
"Sylvie-Anne"	514 096 4638
"Luc"	604 394 6870
"Sofia"	206 384 8697

#### A hash table is a generalization of the simple fixed-size array

• In a fixed size array, you access elements by their index, which is a number between 0 and the size of the array minus 1

```
1 // Declaring a string array
2 String[] someArray = { "514 398 2186", "438 887 1414",
3 "609 234 7564", "555 567 9876" };
4 
5 // Accessing the value at index 2
6 System.out.println( someArray[2] );
7
```

This is called direct addressing, since we can access an element of an array with just one operation

• A hash table uses a similar idea, we want to access a value **directly** using its key

```
1
      // This is not valid java code, just an example
 2
 3
      // We would like to do something like:
 4
 5
       // Accesing the value at index "John"
 б
      String phoneNumber = someHashTable[ "John" ];
 7
 8
       // Accesing the value at index "Anita"
 9
       System.out.println("Anita's phone number is "+someHashTable[ "Anita"] );
10
```

#### To get direct access, a hash table uses a fixed-size array to store data.

We would associate each key to a position in the array, and store the data in the corresponding slot in the array.



# We could try and have one slot in the array for every possible key; i.e. map each possible name to a number in the array

This is not feasible if the number of keys is very big (How many possible names are there?)

Furthermore, we might not even need all of the possible positions in the array to store data

#### Hashing

Instead, we use a fixed-size array with a small size. For example, we could use just 26 slots for all possible names in a phone book.



We need to compute an index for every key. This is called hashing .

# **Hashing function**

A hashing function maps input data of arbitrary size to output data of fixed size; i.e. The set of all possible person names to 26 slots



For a key k, we store its value in the slot given by the hashing function f(k)

We say that f(k) is the hash value of the key k

A good hashing function would give *different* hash values to every key

#### **Hashing function**

But since we are limiting the size of the fixed-size array (the number of slots available) ...



#### .. multiple keys will have the same hash value. This is called a collision

• In our example, the hash function maps the first letter of a name to a number from 0 to 25; representing its position in the alphabet

• Inevitably, as we add more elements to the hash table, multiple elements will have the same same hash value.

Collisions happen when more than two keys have the same hash value



#### We can resolve collisions by

- Increasing the size of the array that holds the content, and finding a *better* hashing function
- Chaining multiple values in the same slot, using a LinkedList

Collisions happen when more than two keys have the same hash value



#### We can resolve collisions by

- Increasing the size of the array that holds the content, and finding a *better* hashing function
- Chaining multiple values in the same slot; e.g. using a LinkedList

#### We can resolve collisions by

• Increasing the size of the array that holds the content, and finding a *better* hashing function

Picking a good hashing function is a good idea: In our example, there are a lot more people with a name starting with J, than there are people with a name starting with Q

But as long as there are more keys than available slots (hash values), there will be collisions.

#### We can resolve collisions by

• Chaining multiple values in the same slot; e.g. using a LinkedList



We lose some of the benefit of direct addressing. But this is: 1) faster than a Linked list on average 2) More flexible than a fixed-size array

# An implementation of a HashTable

See the provided code in the course website

# An exercise for you:

Download the baby-names.csv file from the course website

- Using a Hashtable, count how many times each name appears on the list . See how popular your name is
- Compute the percentage of baby names with a given starting letter; e.g. how what is the percentage of names starting with a J
- Can you come up with a hashing function that maps baby names to 100 slots? Try loading the names in the baby-names.csv file to your Hashtable. Can you change your hashing function so that every slot has more or less the same number of elements?

# Next class: Visual applications with Processing and Java

#### Processing is a programming language based on Java, tailored for visual applications

The objective is to quickly sketch some idea an produce something that other people can understand

- i.e. rapid prototyping of something that an user can see and interact with
- An image is worth a thousand words



Drawing pretty stuff

Processing is a programming language based on Java, tailored for visual applications

The objective is to quickly sketch some idea an produce something that other people can understand

- i.e. rapid prototyping of something that an user can see and interact with
- An image is worth a thousand words



Visualizing data

Processing is a programming language based on Java, tailored for visual applications

The objective is to quickly sketch some idea an produce something that other people can understand

- i.e. rapid prototyping of something that an user can see and interact with
- An image is worth a thousand words



Plotting mathematical functions

Processing is a programming language based on Java, tailored for visual applications

The objective is to quickly sketch some idea an produce something that other people can understand

- i.e. rapid prototyping of something that an user can see and interact with
- An image is worth a thousand words



Simulating Newton's gravitation laws

Processing is a programming language based on Java, tailored for visual applications

The objective is to quickly sketch some idea an produce something that other people can understand

- i.e. rapid prototyping of something that an user can see and interact with
- An image is worth a thousand words



Simulating Artificial Intelligence (Little robots following a path while avoiding each other)

# The Processing Interactive Development Environment (IDE)

Go get it at <a href="https://processing.org/download/">https://processing.org/download/</a>



We will review most of the COMP 202 Java concepts within this Processing window

# Processing

#### Processing is a programming language based on Java

It facilitates building graphical user interfaces (GUI), by abstracting some of the lower level concepts from Java

- You don't have to deal with the Java way of drawing things on the screen (e.g. see code for plotting in A3, or the Country map in A4)
- It hides from the programmer some of the basic constructs from Java
- But you can still use Java code in processing
  - $\circ\,$  Using Java data types ( int, double, String )
  - Importing standard Java libraries ( ArrayList, Hashtable, Math, Random, etc)
  - Creating your own object classes
  - Reading and writing files
  - Handling Exceptions

#### Moving on from our console (text input and output) applications

#### Hello world in Java

```
1 public class HelloWorld{
2 public static void main(){
3 System.out.println("Hello World"); // printing in console
4 }
5 }
6
```

#### Hello world in Processing

println("Hello World"); // printing in console
size(640, 480); // setting up a window that is 640 pixels wide, by 320 pixels tall
text("Hello World", 320, 240); // print hello world in the middle of the window

## Resources

- Exceptions: https://docs.oracle.com/javase/tutorial/essential/exceptions/
- Reading and writing files: http://www.homeandlearn.co.uk/java/read\_a\_textfile\_in\_java.html
- Static and Final: https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
- Processing: <u>http://processing.org</u>