Luis Eduardo Escobar Garcés
A01206734

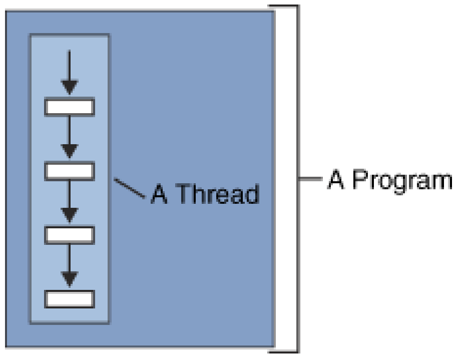
JAVA CONCURRENCY

RABBIT VS TURTLE

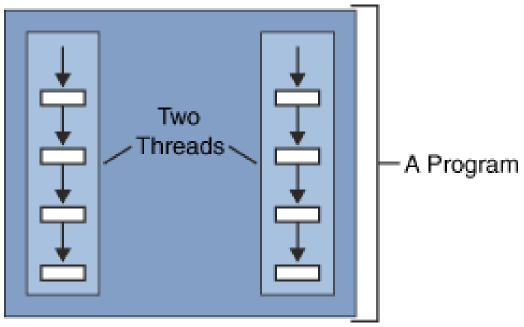
When we access to internet and open a tab and then other tab and so on, you are being part of one of the best programming languages techniques ***concurrency***, when you open each tab, your browser is creating a thread on each one, so if you want, you can view several tabs at the same time!

Definition

A thread can be defined as *"A thread is a single sequential flow of control within a program"* and we can see it as next image.



But in most of cases we use more than 1, because we can have them running at the same time, performing the same task or different tasks, depends on the problem we want to solve, as it shows next image.



Advantages

Reducing time

One of the best advantages of using threads is *time*, we can do tasks in less time as if we do it in a sequential program, for example if we want to discover the password of someone using brute force proving all the combinations of characters, if the password is of a length of 26 characters for this mere example, we will have a total of 26 X 26 X 26 X 26 = 456,976 combinations, if we use a sequential program it would take more time to go over each of the combination, but what if we use 8 threads concurrently, each thread would do 57,122 combinations each, so we would find the password in less time.

Interactivity

Concurrency allows the user to interact with several applications at the same time but not necessarily simultaneously, examples of interactivity systems are like browser, watching a video and stopping it, games.

Use of resources

Multicore architectures processors can be used with a better use, for heavy calculations, games, simulations, applications, etc.

Threads and Processes

There are differences between threads and processes, as it can be seen in the next table.

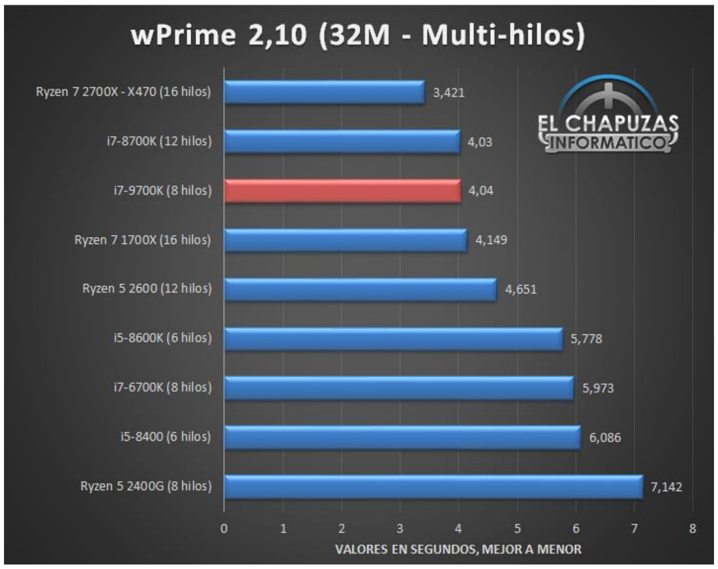
Process	Thread
A process is not inside a thread	A thread is inside a process
Cannot communicate with other process	Can communicate with other thread or modify
Requires more resources	Requires less resources
Heavyweight context	Lightweight context
Difficult to create	Easier to create

As we can see, both are different, processes are seen in operative systems, because they communicate with it, and threads we can see them in high or medium level applications as video games, images rendering, browsers, applications with communication between user and client, etc.

Architectures

Threads most of cases are used in multicore architectures, because they are better used. When we program with threads, each is assigned to a core, and each core allows a certain number of threads.

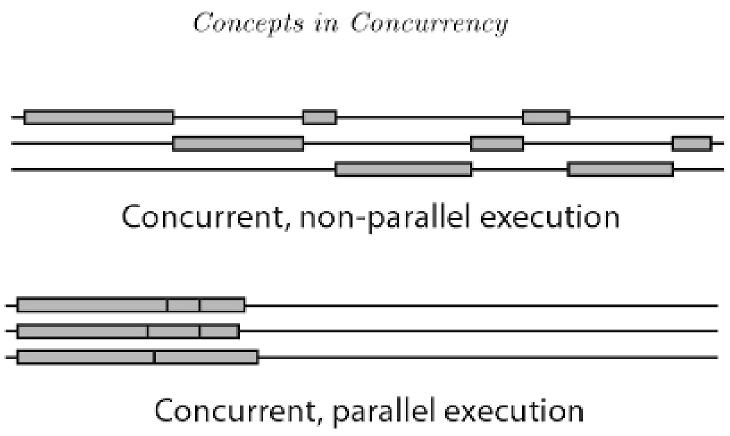
Threads applications performance depends highly in your architecture, so if you have a Ryzen 7 2700X – X470 of 16 threads per core you certainly will have a better experience and performance, as we can see in the next image, where it shows time performance from better to worse.



Parallelism and concurrency

There is a difference between parallelism and concurrency, in concurrency we could create several threads running at the same time in different cores, but not necessarily they are running in a simultaneous way, parallelism can run several threads in multiple processes in a simultaneous way at the same time.

You can have a single thread running on a single core with concurrency, but with parallelism not, we would have it in several processes.


Concepts in Concurrency
Concurrent, non-parallel execution
Concurrent, parallel execution

JAVA

Java is objected-oriented language programming very important throughout the world, it is present from mobile applications, desktop applications, web development and databases, Java is used in more than 3 billion devices, as we can observe in the next Java installation window.

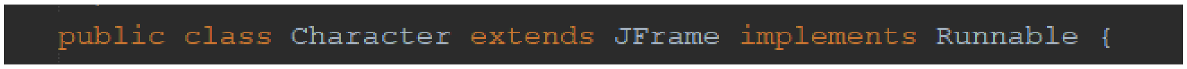
Concurrency in JAVA

Concurrency is implemented in Java, so we can make use of this feature to create programs that they would be better if they are implemented with concurrency programming.

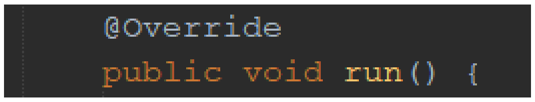
There are two ways to make a task run concurrently with another: create a new class as a subclass of the Thread class or declare a class and implement the Runnable interface. We are going to implement the Runnable interface in another class.

Runnable interface

We are going to implement run method in the class that it is going to do the instructions that we decide, so when we implement runnable interface, we will need to add the method run().

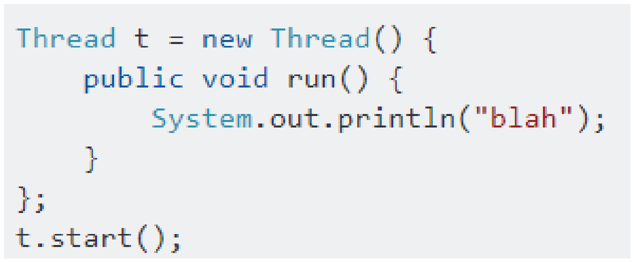


```java
public class Character extends JFrame implements Runnable {
```



```java
    @Override
    public void run() {
```

We also can implement the run() method with an anonymous class as next example.



```java
Thread t = new Thread() {
    public void run() {
        System.out.println("blah");
    }
};
t.start();
```

As we can see in the image, we create a variable t of type Thread, and then we create our anonymous class.

In the method run() it is going to be all the tasks that are going to be executed by the thread or threads, but how do we start our thread?

There are 4 methods that are the most common when a thread variable is created. One of them is t.start() as we can see in the image above.

Void Start() : As the name says, it starts the execution of our thread, we are going to do an start() with each thread that we are creating. Immediately when it is started, the method run() starts running.

Void sleep(): This method puts the thread to sleep a certain time, for example sleep(1000), 1000 milliseconds.

Void join(): This method puts the thread in a waiting state , and reminds in a waiting state until the referenced thread terminates.

Void interrupt(): An interrupt is an indication to a thread that it should stop what it is doing and do something else.

Luis Eduardo Escobar Garcés
A01206734

RABBIT VS TURTLE RACE SIMULATION

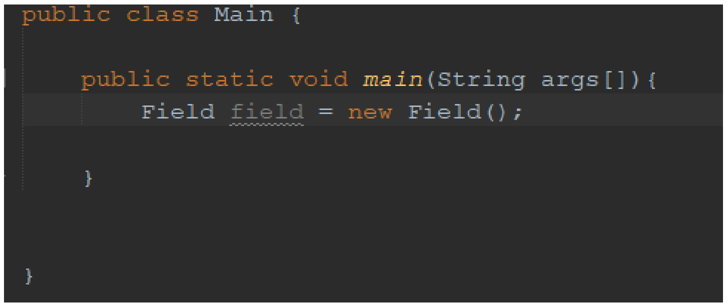
As we saw previously a briefly introduction of how concurrency in Java works, I am going to explain you about a race simulation between a rabbit and a turtle using concurrency, the purpose of this simulation it is to explain in a clear and easy way, how a basic concurrency program works.

Scenery

The scenery is a Rabbit and a Turtle that are each one threads, that are running concurrently, and it ends until one of the them wins, or it is a tie.

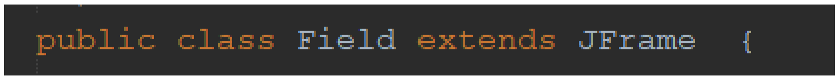
The project is made of 3 clases, Main, Field, Character.

In Main class I only call Field class.



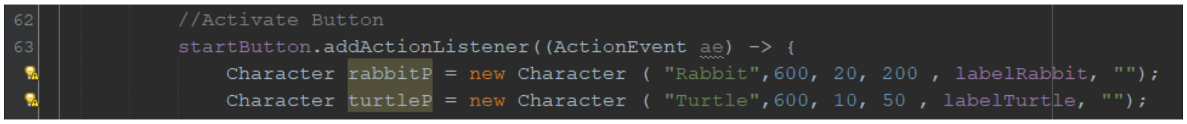
```
public class Main {

    public static void main(String args[]){
        Field field = new Field();

    }


}
```

Class Character, is extended to JFrame, to use JLabels and display our images in the window.



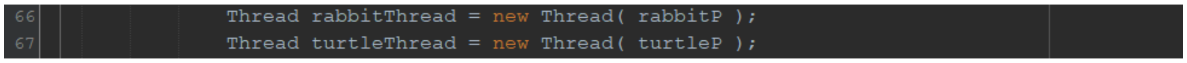
```
public class Field extends JFrame  {
```

But let’s go to the part that we are interested, in the concurrency part. As we will see in the next image, I create a button that it is going to start our race simulation. First, I instantiate our class Character and create it with different parameters for rabbitP, and for turtleP. The last parameter is the “win condition” string, it is only important in the tie button, for now ignore it.



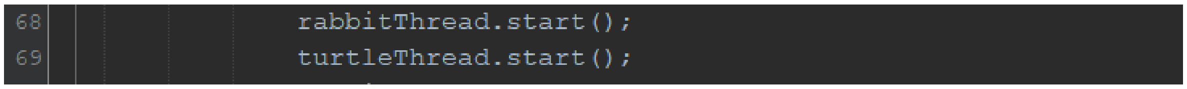
```
        //Activate Button
        startButton.addActionListener((ActionEvent ae) -> {
            Character rabbitP = new Character ( "Rabbit",600, 20, 200 , labelRabbit, "");
            Character turtleP = new Character ( "Turtle",600, 10, 50 , labelTurtle, "");
```

Then when we instantiate out class, I create threads as, rabbitThread and turtleThread, and then we pass it the objects previously created. It is necessary to pass them the objects, because otherwise our thread it is not going to access that information.



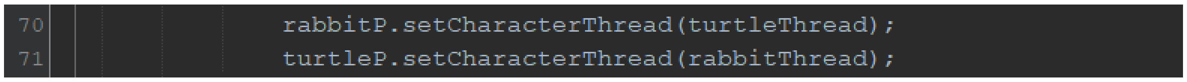
```
            Thread rabbitThread = new Thread( rabbitP );
            Thread turtleThread = new Thread( turtleP );
```

Then we start our thread with start(), this method it is going to call run() method that is inside Character class.



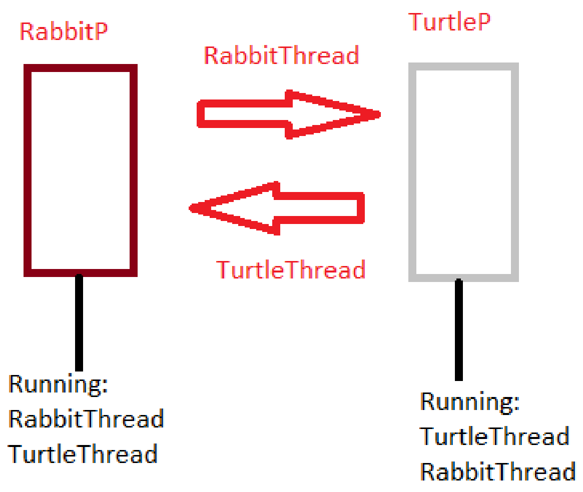
```
            rabbitThread.start();
            turtleThread.start();
```

And finally it is important to set the thread that we are creating to each instance of the class, using the method setCharacterThread() so they could know the data of the other thread that it is running on them. We pass to rabbitP the turtleThread because rabbitP needs to know when the turtleThread it is going to end, and so with turtleP, it needs to know when rabbitThread it is going to end the race.

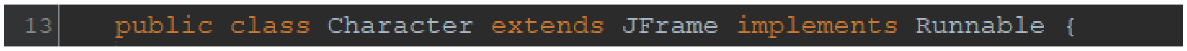


```
            rabbitP.setCharacterThread(turtleThread);
            turtleP.setCharacterThread(rabbitThread);
```

It is going to be running as the image behind.

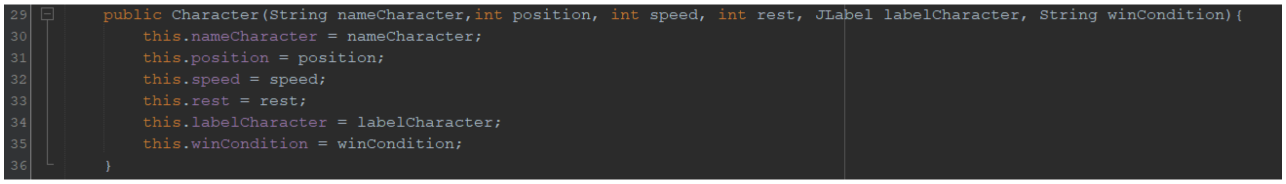


In our class character that it is going to do the operations, we implement Runnable method and create our run method.



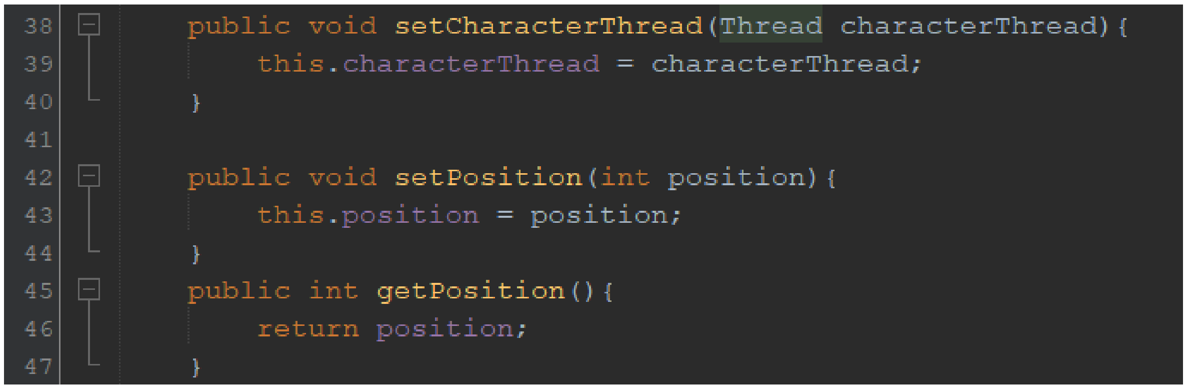
```
public class Character extends JFrame implements Runnable {
```

Next, we initialize our nondefault constructor as follows.



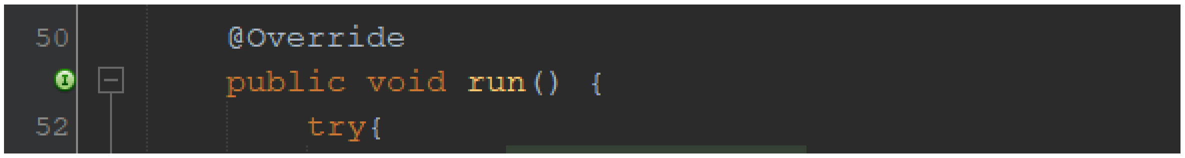
```
    public Character(String nameCharacter,int position, int speed, int rest, JLabel labelCharacter, String winCondition){
        this.nameCharacter = nameCharacter;
        this.position = position;
        this.speed = speed;
        this.rest = rest;
        this.labelCharacter = labelCharacter;
        this.winCondition = winCondition;
    }
```

We are going to declare our setters and getters, for setting the thread that we are receiving, and position of our characters.

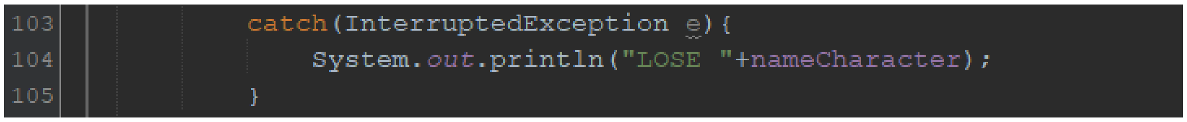


```java
    public void setCharacterThread(Thread characterThread){
        this.characterThread = characterThread;
    }

    public void setPosition(int position){
        this.position = position;
    }
    public int getPosition(){
        return position;
    }
```

Now the part that we are waiting for, run() method, we create a try catch inside, because when one of the threads get interrupted, the InterruptedException will be activated. So, for example if RabbitThread wins, TurtleThread will be interrupted and will be catch in the InterruptedException as the looser.

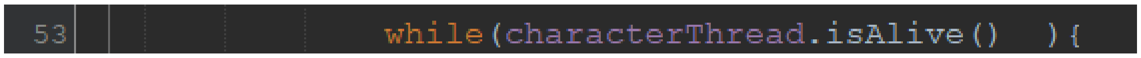


```java
    @Override
    public void run() {
        try{
```



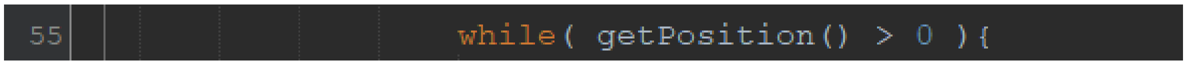
```java
        catch(InterruptedException e){
            System.out.println("LOSE "+nameCharacter);
        }
```

Inside of our try catch we have a while condition that will be true until one of both threads get interrupted, we use isAlive() method to verify, it returns true if it is alive, or false if it is not. As 2 threads will be running concurrently in Character object, it is going to check concurrently if TurtleThread.isAlive() or RabbitThread.isAlive().



```java
            while(characterThread.isAlive()   ){
```

The next while condition we have is until the current thread position is greater than 0, it is going to stop it. We chose 0 because Rabbit and Turtle labels are going up and the position of the window of y axe is 0.



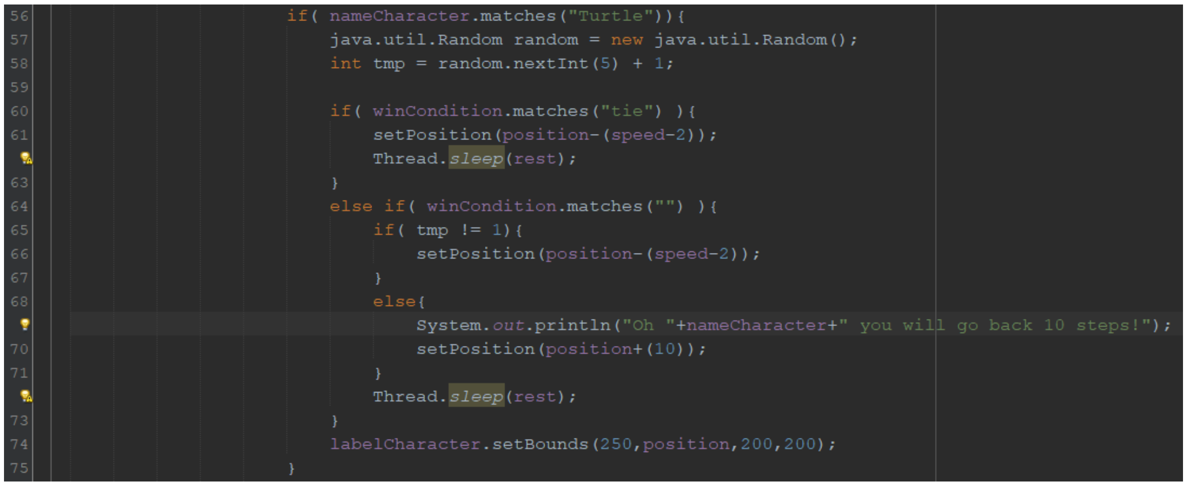
```java
                while( getPosition() > 0 ){
```

Next part of the code, are conditions to give rules to advance for characters, the rules are the next ones:

Rules:

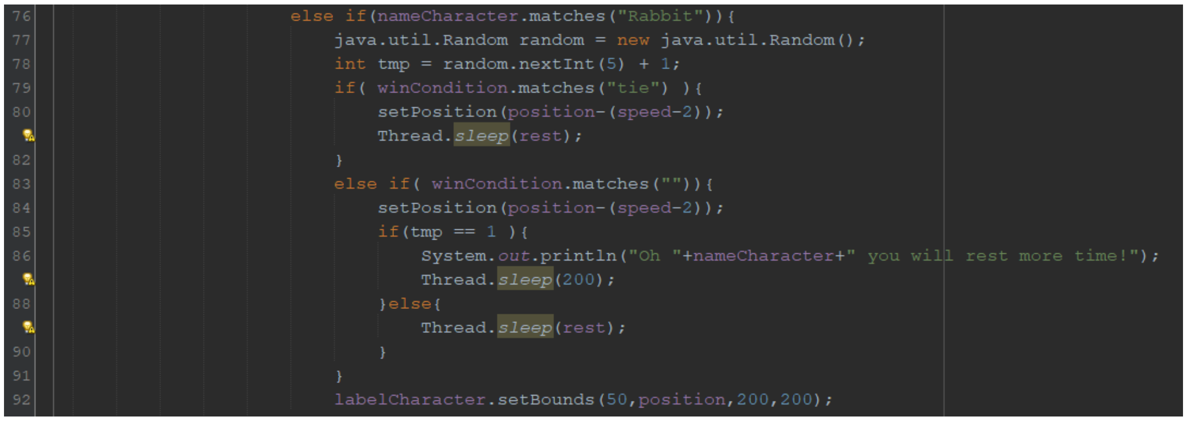
Turtle	**Rabbit**
Between a random number of 1 and 5. If the number is not 1 it is going to advance normally. If the random number is equal to 1 turtle will go back 10 steps And after that it will rest the rest value given when the thread was created	Between a random number of 1 and 5. If the random number is equal to 1, the RabbitLabel will rest 200 If the random number is different from 1, just would rest the given time when the thread was created

In code will be something like this for Turtle



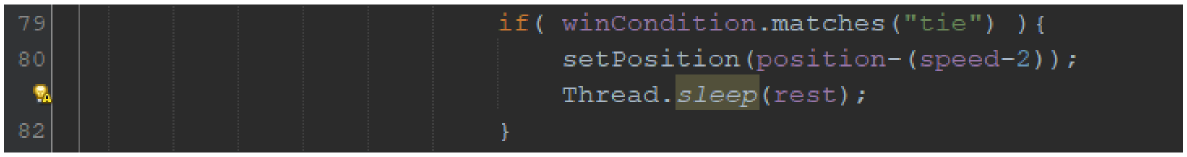
```
if( nameCharacter.matches("Turtle")){
    java.util.Random random = new java.util.Random();
    int tmp = random.nextInt(5) + 1;

    if( winCondition.matches("tie") ){
        setPosition(position-(speed-2));
        Thread.sleep(rest);
    }
    else if( winCondition.matches("") ){
        if( tmp != 1){
            setPosition(position-(speed-2));
        }
        else{
            System.out.println("Oh "+nameCharacter+" you will go back 10 steps!");
            setPosition(position+(10));
        }
        Thread.sleep(rest);
    }
    labelCharacter.setBounds(250,position,200,200);
}
```

And like this for Rabbit



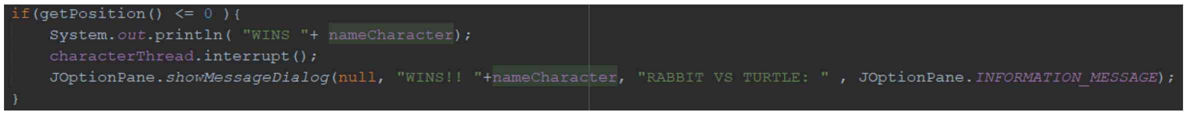
```
else if(nameCharacter.matches("Rabbit")){
    java.util.Random random = new java.util.Random();
    int tmp = random.nextInt(5) + 1;
    if( winCondition.matches("tie") ){
        setPosition(position-(speed-2));
        Thread.sleep(rest);
    }
    else if( winCondition.matches("")){
        setPosition(position-(speed-2));
        if(tmp == 1 ){
            System.out.println("Oh "+nameCharacter+" you will rest more time!");
            Thread.sleep(200);
        }else{
            Thread.sleep(rest);
        }
    }
    labelCharacter.setBounds(50,position,200,200);
```

You are probably wondering why we have a condition like the next one. This condition has it for both threads, and it is just to force a TIE.



```
                    if( winCondition.matches("tie") ){
                        setPosition(position-(speed-2));
                        Thread.sleep(rest);
                    }
```

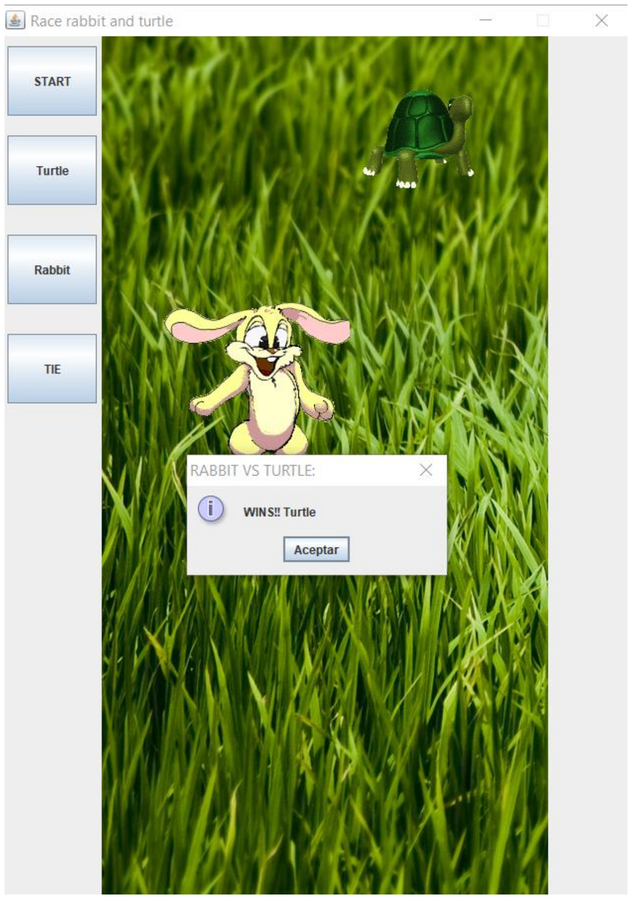
After the previously rules that we saw, we have our win condition, like this. The first thread that enters in this if, if its position is less or equal than 0, it means that it has reached the goal.



```
if(getPosition() <= 0 ){
    System.out.println( "WINS "+ nameCharacter);
    characterThread.interrupt();
    JOptionPane.showMessageDialog(null, "WINS!! "+nameCharacter, "RABBIT VS TURTLE: " , JOptionPane.INFORMATION_MESSAGE);
}
```

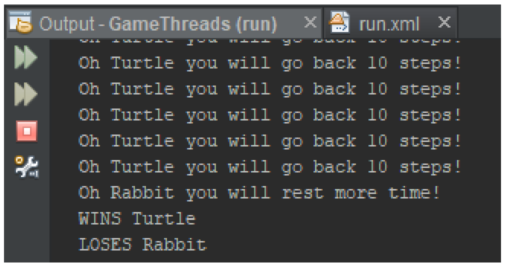
The thread that first reached position 0, it is going to be the winner, so then the thread that has not finished yet, gets interrupted, and is being catched by InterruptionException. And we can see the next two results.

If turtle wins



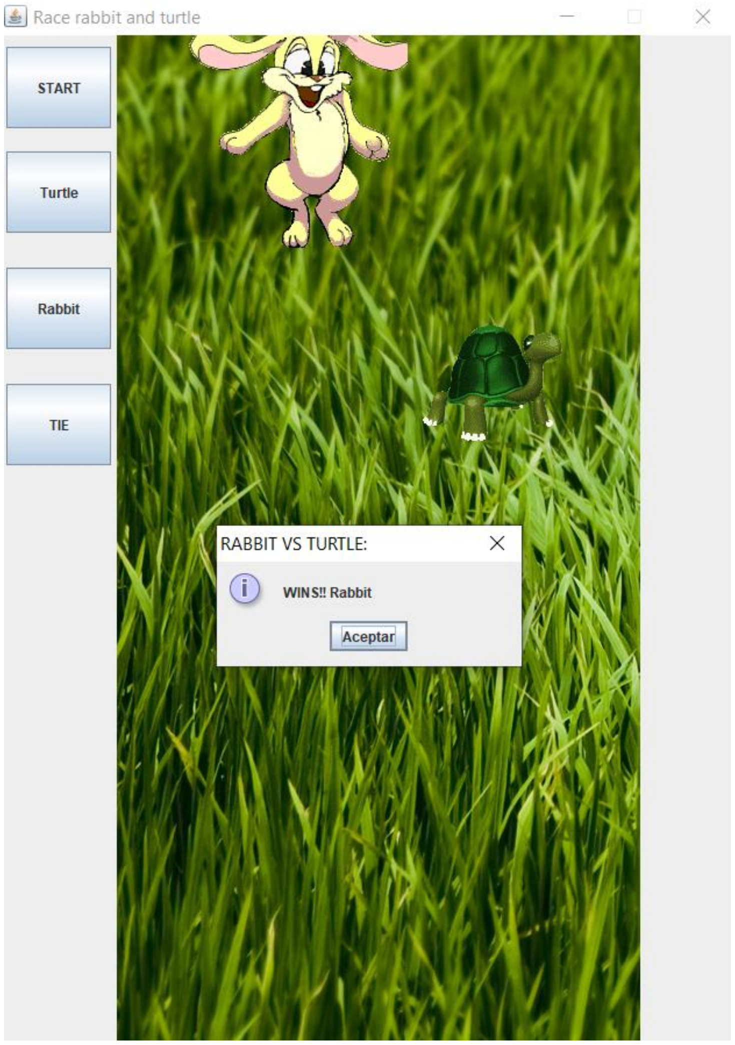
RABBIT VS TURTLE:

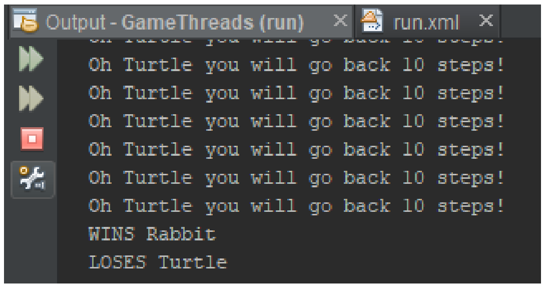
WINS!! Turtle



```
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Rabbit you will rest more time!
WINS Turtle
LOSES Rabbit
```

If Rabbit wins:



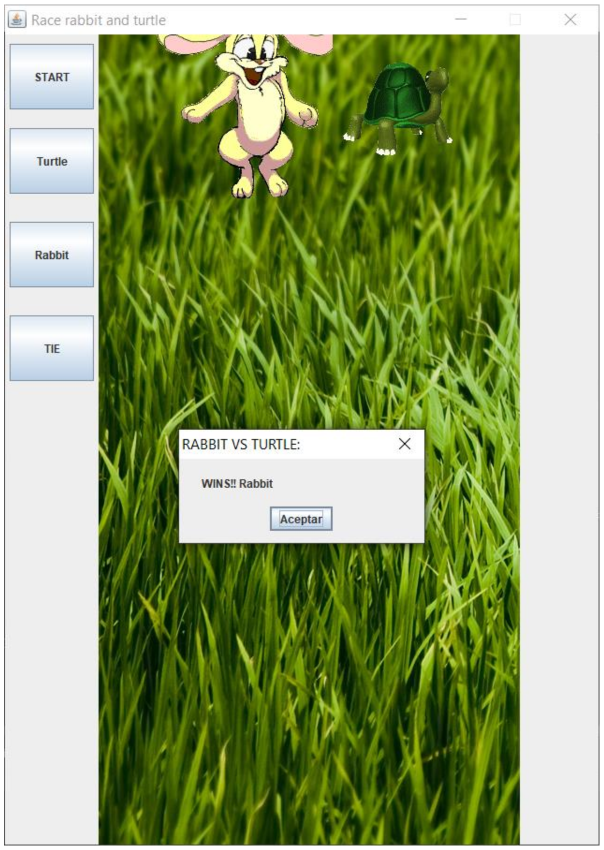


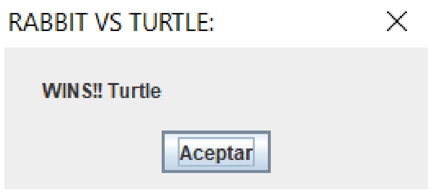
```
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
Oh Turtle you will go back 10 steps!
WINS Rabbit
LOSES Turtle
```

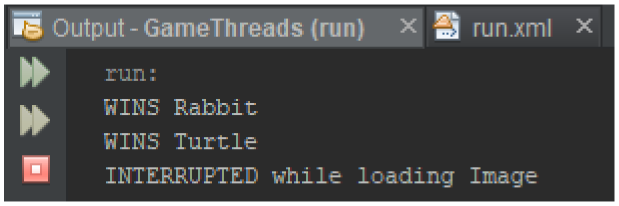
But what happens if we have a TIE?

When both threads reach a position less or equal than 0, as we saw back in theory, concurrency does not always run simultaneous at the same time, so both threads reach at the same time the condition and none of them are being interrupted, but it can be the case that one is going to arrive first, so we can have issues with outputs. This is called race condition, when several threads try to access a shared variable or method.

Both message dialogs are being displayed and also in console

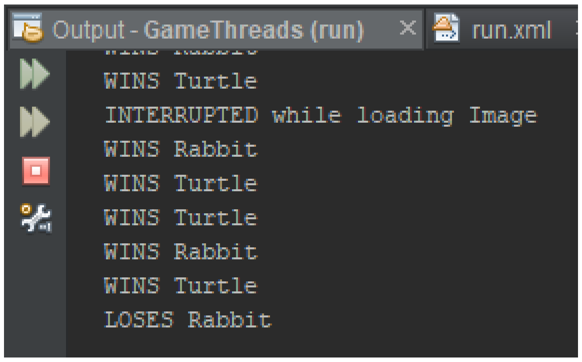






```
run:
WINS Rabbit
WINS Turtle
INTERRUPTED while loading Image
```

But also it can happen irregularities if we run it again. It can happen that says that either turtle or rabbit wins, we ran the program several times as we can see several outputs.



```
WINS Turtle
INTERRUPTED while loading Image
WINS Rabbit
WINS Turtle
WINS Turtle
WINS Rabbit
WINS Turtle
LOSES Rabbit
```

Conclusions:

Concurrency is a very important language paradigm in industry, and it is present in a lot of systems, it can be helpful in multicores architectures to solve problems in an efficient way, providing user sense of interaction with multiple programs, or faster performance. Besides this, if it is not programmed properly it could lead to the think of “nothing bad is going to happen” and also take into account that not necessarily all concurrent programs are faster than sequential ones even, if you have multiple processors. It is important to consider this and use them properly.

CODE:

This is the link to the code on GitHub

https://github.com/LuisEduardoEscobarGarces/Programming-languages

References:

- ITUU. (2005). Threads. 28/05/2020, de ITUU Sitio web: https://www.it.uu.se/edu/course/homepage/games/Threads/threads.pdf
- Krishnabhatia. (2010). What are the advantages of concurrency? 28/05/2020, de Geek for geeks Sitio web: https://practice.geeksforgeeks.org/problems/what-are-the-advantages-of-concurrency
- Carlos Varela. (2007). Concurrent Programming. 28/05/2020, de CSRPI Sitio web: https://www.cs.rpi.edu/academics/courses/spring07/dci/SALSA-Concurrency.pdf
- Lithme. (2018). Difference Between Process and Thread. 28/05/2020, de Pediaa Sitio web: https://pediaa.com/difference-between-process-and-thread/
- Iván Martínez. (2018). Review: Intel Core i7-9700K (Exclusiva) Read more https://elchapuzasinformatico.com/2018/09/intel-core-i7-9700k-review/. 29/05/2020, de El chapuzas informatico Sitio web: Review: Intel Core i7-9700K (Exclusiva) Read more https://elchapuzasinformatico.com/2018/09/intel-core-i7-9700k-review/
- Baeldung. (2019). The Thread.join() Method in Java. 29/05/2020, de Beldung Sitio web: https://www.baeldung.com/java-thread-join
- RichieHindle. (2009). What is the difference between concurrency and parallelism? 28/05/2020, de StackOverflow Sitio web: https://stackoverflow.com/questions/1050222/what-is-the-difference-between-concurrency-and-parallelism
- ORACLE. (2019). The Java™ Tutorials. 28/05/2020, de ORACLE Sitio web: https://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
- ProfesoresELO. (2008). Hilos (Threads) en Java. 28/05/2020, de ProfesoresELO Sitio web: http://profesores.elo.utfsm.cl/~agv/elo330/2s10/lectures/Java/threads/JavaThreads.html