# Tutorial - Week11 5COSC019W – Object Oriented Programming – Java

**Multithreading**

**5-6 December**

1) **Extending thread Class**

You will create and start a thread execution by writing a class that extends Thread class. The thread you will implement has to write in the screen 10 times the name of the thread that is executed.

a) Create a class by extending Thread class and override run() method

```
public class PrintNameThread extends Thread{

    // constructor
    PrintNameThread(String threadName){
        super (threadName);
    }

    // run method
    public void run(){
        //print 10 times the name
        for(int i = 0; i < 10;  i++){
            System.out.println("Thread name: " + this.getName());
        }
    }
}
```

Override the run() method of the Thread class.

This method gets executed when start() method is invoked

b) Create a thread in the main() and call the start method()

```
public class ExtendThread {

    public static void main(String[] args) {

        // create a thread A
        PrintNameThread threadA = new PrintNameThread("A");
        threadA.start();
    }

}
```

Create an object instance of the class that is subclass of Thread class

Start the thread by invoking the start() method

c) Instantiate other 2 threads called “B” and “C” and start them. Check what will be displayed in the screen. If you run several times you will see that the sequence of the displayed names will change. Why?

2) **Implement Runnable interface**

In this exercise you will solve the previus task implementing the interface Runnable.

a) Create a class that implements the interface Runnable and override run() method:

```
public class PrintNameRunnable implements Runnable{

    //intsnace variable
    String nameThread;

    //constructor
    public PrintNameRunnable(String nameThread){
        this.nameThread = nameThread;
    }
```

Implement run() method defined in the Runnable interface

```
    //run method
    public void run(){
        //print 10 times the name
        for(int i = 0; i < 10;  i++){
            System.out.println("Thread name: " + nameThread);
        }
    }
}
```

b) Start a thread with the names "A" using the class PrintNameRunnable.

```
public class RunnableThread {

    public static void main(String[] args) {

        // Create the object PrintNameRunnable
        PrintNameRunnable printA = new PrintNameRunnable("A");

        // Create the Thread object
        Thread threadA = new Thread(printA);

        // start the thread
        threadA.start();
    }
}
```

Note: the start() method has to be invoked after an object PrintNameRunnable has been instantiated

d) Start also the other two threads to print "B" and "C"

## Consumer and Producer

**Problem**: Two threads, the producer and the consumer, sharing a common fixed-size buffer.

- The producer generate a piece of data and put into the buffer.
- The consumer is consuming the data from the same buffer simultaneously.
- Notes:
  - The producer should wait when it tries to put the new data in the buffer until there is at least one free slot in the buffer.
  - The consumer should stop consuming if the buffer is empty.

1) Implement a class `MessageQueue`, which hold a buffer of Strings with a defined size. Implement two classes: `Consumer` and `Producer`. The Consumer object will read the string from the buffer while the Produces will write a String into the buffer. You need to consider that the Producer can write only if there is space in the buffer and that the Consumer should stop consuming if the buffer is empty.
   You can find the structure of the code below. Try to fill the methods with appropriate code. Remember to use wait(), notifyAll() for the synchronisation of the threads.

MessageQueue.java:

```
public class MessageQueue {

    //the size of the buffer
    private int bufferSize;

    //the buffer list of the message, assuming the string message format
    private List<String> buffer = new ArrayList<String>();

    //construct the message queue with given buffer size
    public MessageQueue(int bufferSize){
            // here your code
    }

    //check whether the buffer is full
```

```
    public synchronized boolean isFull() {
            // here your code    }

    //check whether the buffer is empty
    public synchronized boolean isEmpty() {
            // here your code
    }

    //put an income message into the queue, called by message producer
    public synchronized void put(String message) {
            // here your code
   }

    //get a message from the queue, called by the message consumer
    public synchronized String get(){
            // here your code
}
```

Consumer.java

```
public class Consumer extends Thread{

    private MessageQueue queue = null;

      public Consumer(MessageQueue queue){
            this.queue = queue;
      }

      public void run(){
            for(int i=0;i<10;i++){
                  System.out.println("Consumer downloads " + queue.get()+ " from the
                  queue");
            }
      }
}
```

Producer.java

```
public class Producer extends Thread {

    private static int count = 0;
    private MessageQueue queue = null;

public Producer(MessageQueue queue){
      this.queue = queue;
}

public void run(){
      for(int i=0;i<10;i++){
            queue.put(generateMessage());
      }
}

private synchronized String generateMessage(){
      String msg = " Message numebr " +count;
      count ++;
      return msg;
}
}
```