Module Title: Distribute and Cloud Systems Programming (5CS022)

Subject Title: Workshop- 4

Student Name: Nayan Raj Khanal

Student Code: 2227486

Instructor: Mr. Prabin Sapkota

Submitted on: 26.03.2023

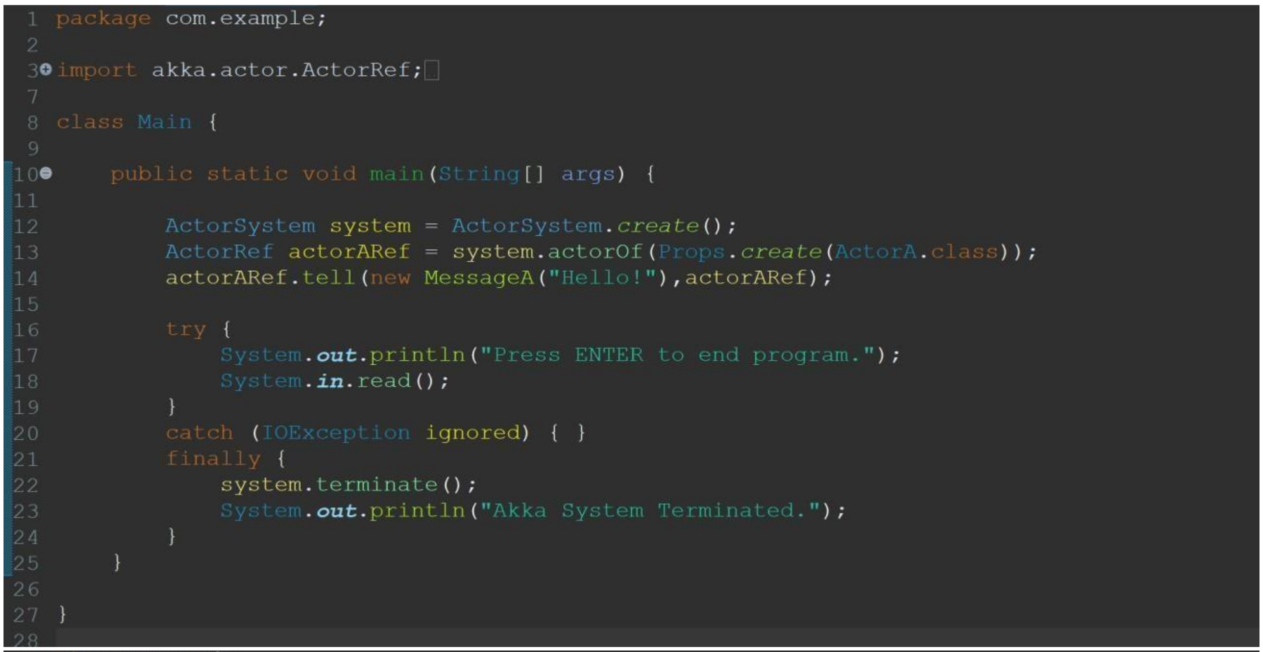
University of Wolverhampton

School of Mathematics and Computer Science

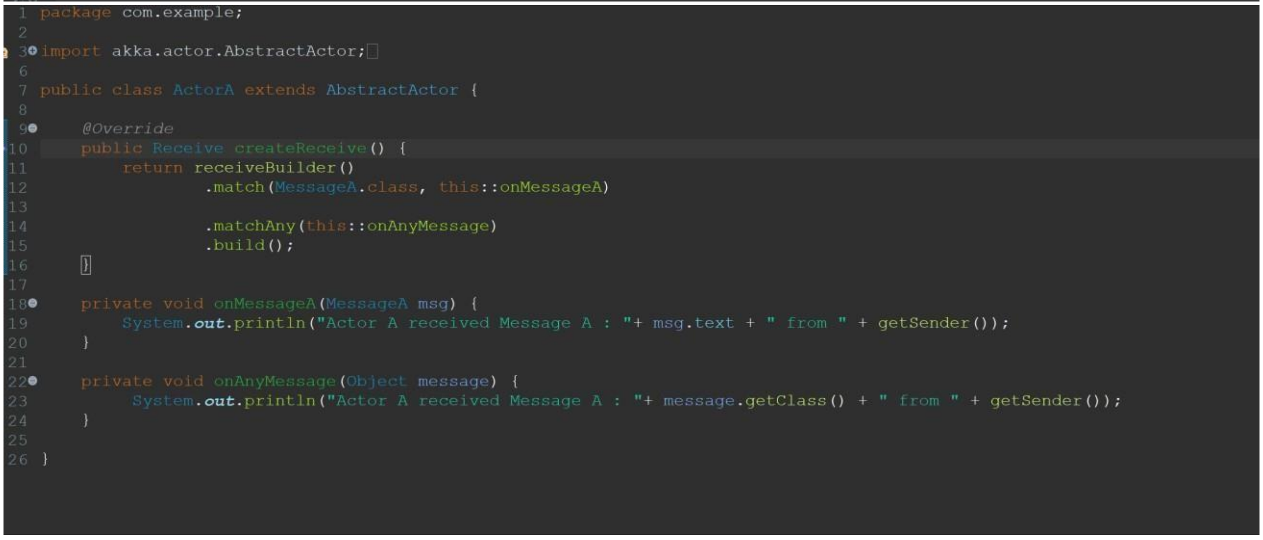
5CS022 Distribute and Cloud Systems Programming

Workshop 4 The Akka Framework Part 2

1. The Actor class ActorA in the sample Akka program currently responds to only one message object – MessageA. Modify the createReceive() method so that it will also respond to any other message and print out the message on the standard output.

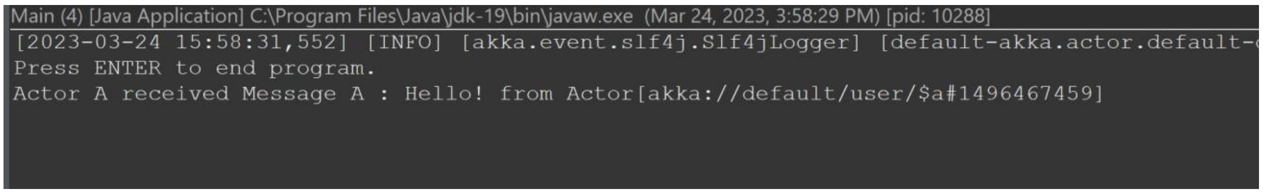


```java
package com.example;

import akka.actor.ActorRef;

class Main {

    public static void main(String[] args) {

        ActorSystem system = ActorSystem.create();
        ActorRef actorARef = system.actorOf(Props.create(ActorA.class));
        actorARef.tell(new MessageA("Hello!"),actorARef);

        try {
            System.out.println("Press ENTER to end program.");
            System.in.read();
        }
        catch (IOException ignored) { }
        finally {
            system.terminate();
            System.out.println("Akka System Terminated.");
        }
    }

}
```



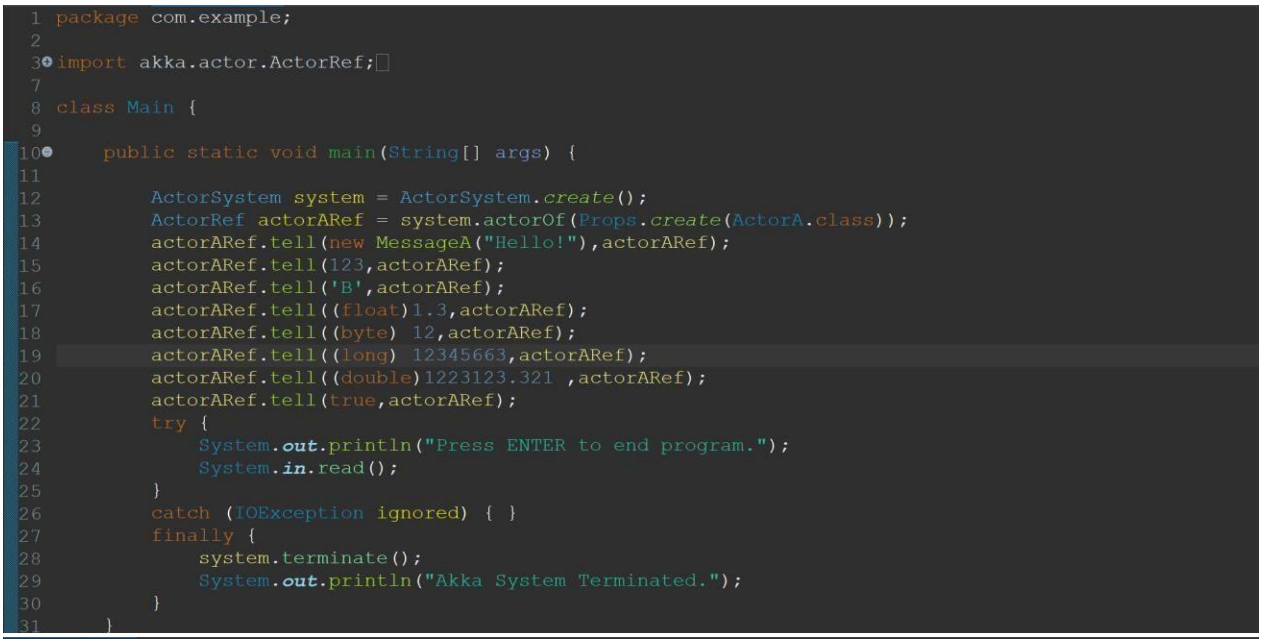
```java
package com.example;

import akka.actor.AbstractActor;

public class ActorA extends AbstractActor {

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(MessageA.class, this::onMessageA)

                .matchAny(this::onAnyMessage)
                .build();
    }

    private void onMessageA(MessageA msg) {
        System.out.println("Actor A received Message A : "+ msg.text + " from " + getSender());
    }

    private void onAnyMessage(Object message) {
         System.out.println("Actor A received Message A : "+ message.getClass() + " from " + getSender());
    }

}
```

Output:

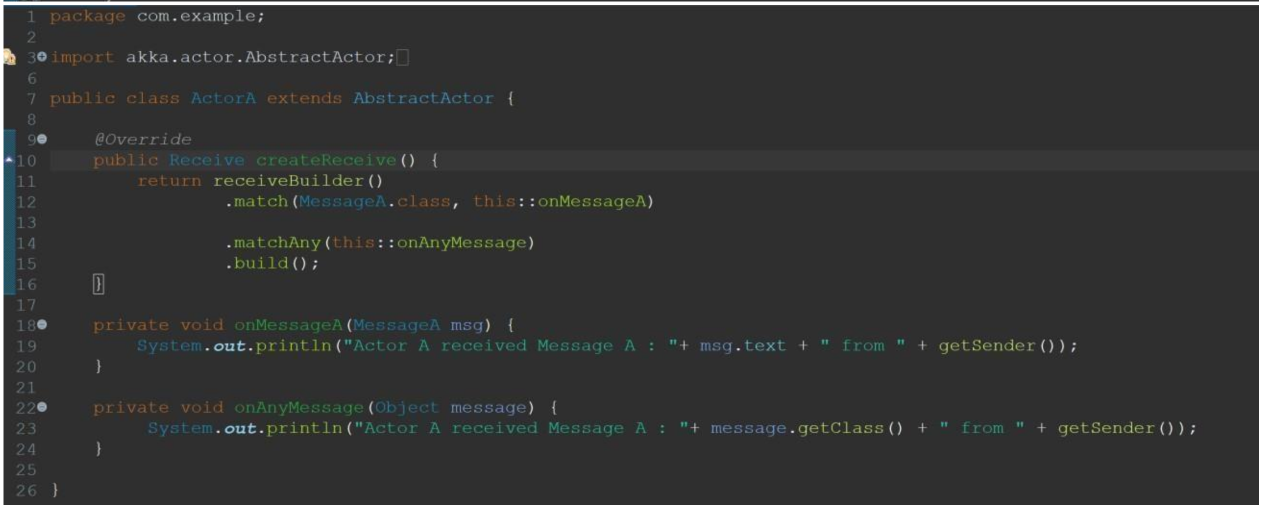


```
Main (4) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (Mar 24, 2023, 3:58:29 PM) [pid: 10288]
[2023-03-24 15:58:31,552] [INFO] [akka.event.slf4j.Slf4jLogger] [default-akka.actor.default-
Press ENTER to end program.
Actor A received Message A : Hello! from Actor[akka://default/user/$a#1496467459]
```

2. Every message in Akka is a Java object. However, not all messages need to have a custom Java class created for them. Convert the sample program so that it can respond to messages that contain the primitive data types such as byte, short, int, long, float, double, boolean, and char, without having to create custom Java classes.

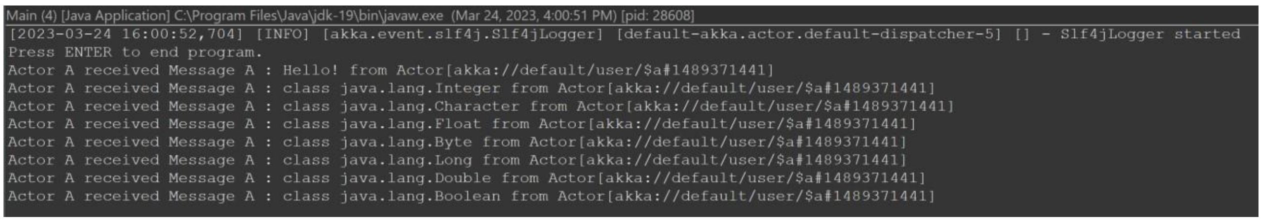


```java
package com.example;

import akka.actor.ActorRef;

class Main {

    public static void main(String[] args) {

        ActorSystem system = ActorSystem.create();
        ActorRef actorARef = system.actorOf(Props.create(ActorA.class));
        actorARef.tell(new MessageA("Hello!"),actorARef);
        actorARef.tell(123,actorARef);
        actorARef.tell('B',actorARef);
        actorARef.tell((float)1.3,actorARef);
        actorARef.tell((byte) 12,actorARef);
        actorARef.tell((long) 12345663,actorARef);
        actorARef.tell((double)1223123.321 ,actorARef);
        actorARef.tell(true,actorARef);
        try {
            System.out.println("Press ENTER to end program.");
            System.in.read();
        }
        catch (IOException ignored) { }
        finally {
            system.terminate();
            System.out.println("Akka System Terminated.");
        }
    }
```



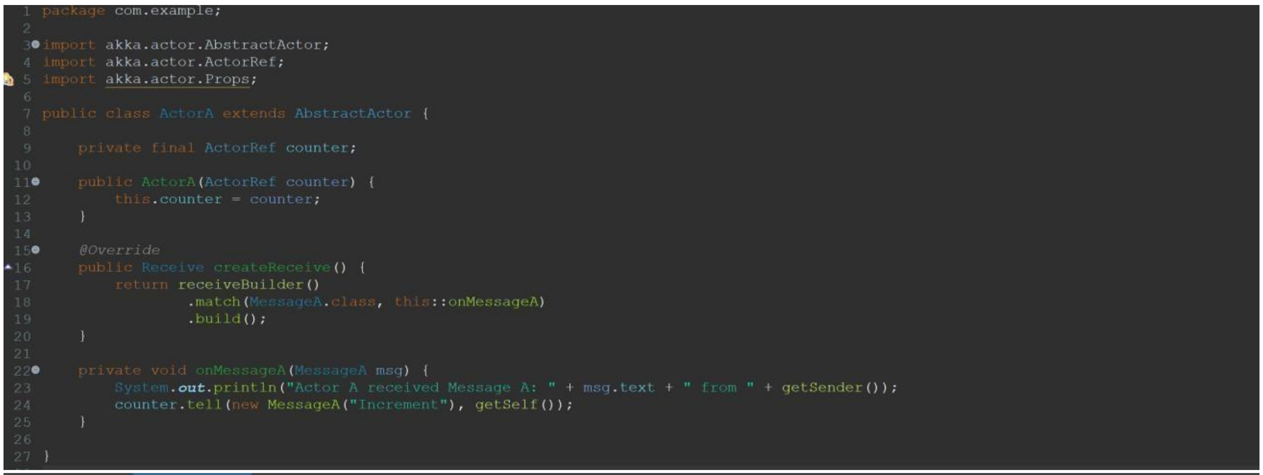
```java
package com.example;

import akka.actor.AbstractActor;

public class ActorA extends AbstractActor {

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(MessageA.class, this::onMessageA)

                .matchAny(this::onAnyMessage)
                .build();
    }

    private void onMessageA(MessageA msg) {
        System.out.println("Actor A received Message A : "+ msg.text + " from " + getSender());
    }

    private void onAnyMessage(Object message) {
         System.out.println("Actor A received Message A : "+ message.getClass() + " from " + getSender());
    }

}
```

Output:

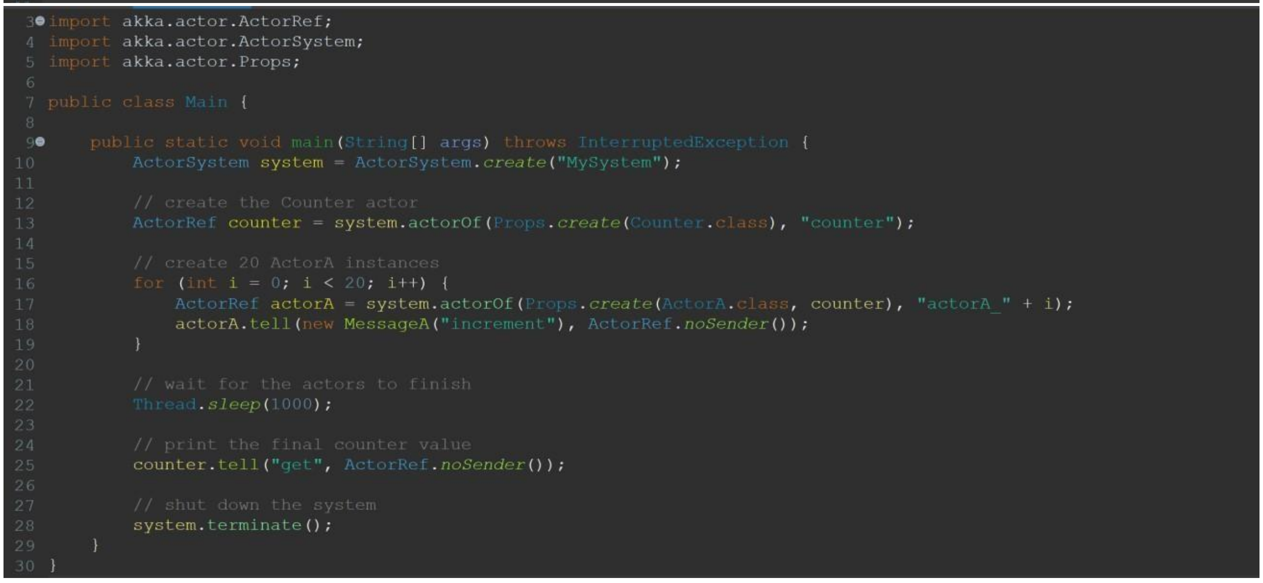


```
Main (4) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (Mar 24, 2023, 4:00:51 PM) [pid: 28608]
[2023-03-24 16:00:52,704] [INFO] [akka.event.slf4j.Slf4jLogger] [default-akka.actor.default-dispatcher-5] [] - Slf4jLogger started
Press ENTER to end program.
Actor A received Message A : Hello! from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Integer from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Character from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Float from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Byte from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Long from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Double from Actor[akka://default/user/$a#1489371441]
Actor A received Message A : class java.lang.Boolean from Actor[akka://default/user/$a#1489371441]
```

3. In standard multithreading programs (for example Pthread programs), shared-resource contention (e.g., global variables) can be an issue and requires the use of mutexes and critical sections. Demonstrate that with Akka Actor, this is not an issue, by creating a "Counter" Actor class to keep track of a global counter, and lots of instances (20) of "ActorA" objects to send messages to "Counter" to increment the global counter.

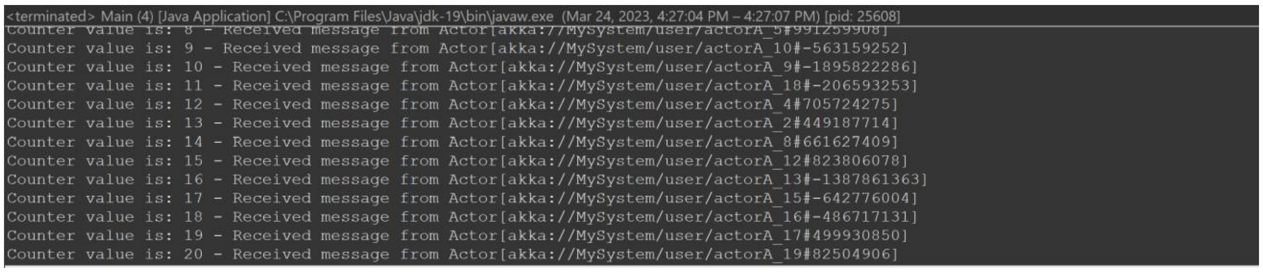


```java
package com.example;

import akka.actor.AbstractActor;
import akka.actor.ActorRef;
import akka.actor.Props;

public class ActorA extends AbstractActor {

    private final ActorRef counter;

    public ActorA(ActorRef counter) {
        this.counter = counter;
    }

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(MessageA.class, this::onMessageA)
                .build();
    }

    private void onMessageA(MessageA msg) {
        System.out.println("Actor A received Message A: " + msg.text + " from " + getSender());
        counter.tell(new MessageA("Increment"), getSelf());
    }

}
```



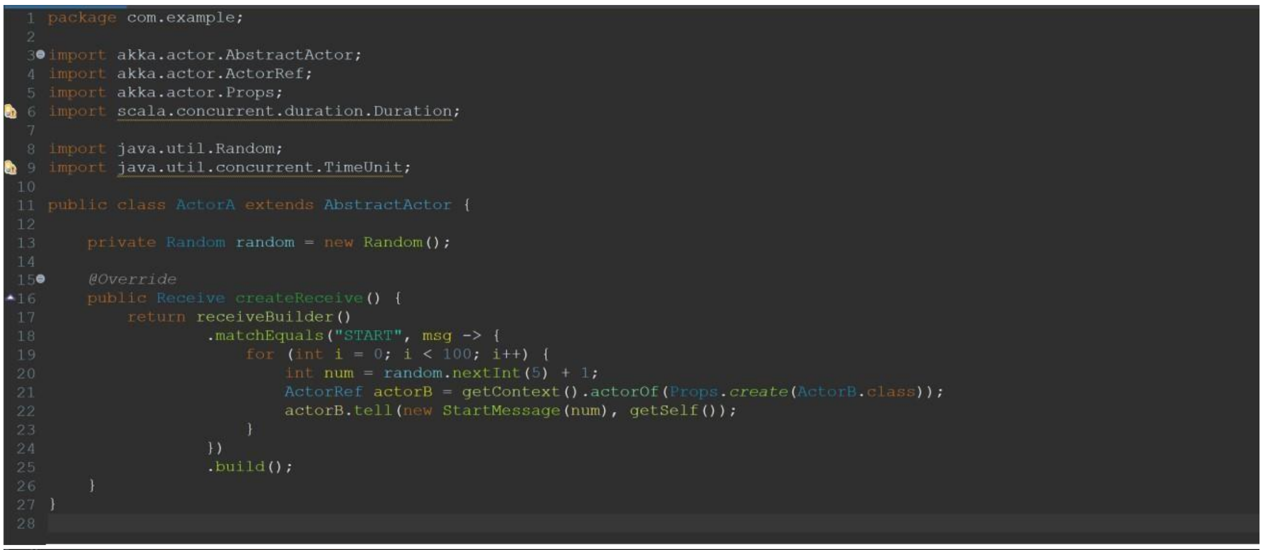
```java
import akka.actor.ActorRef;
import akka.actor.ActorSystem;
import akka.actor.Props;

public class Main {

    public static void main(String[] args) throws InterruptedException {
        ActorSystem system = ActorSystem.create("MySystem");

        // create the Counter actor
        ActorRef counter = system.actorOf(Props.create(Counter.class), "counter");

        // create 20 ActorA instances
        for (int i = 0; i < 20; i++) {
            ActorRef actorA = system.actorOf(Props.create(ActorA.class, counter), "actorA_" + i);
            actorA.tell(new MessageA("increment"), ActorRef.noSender());
        }

        // wait for the actors to finish
        Thread.sleep(1000);

        // print the final counter value
        counter.tell("get", ActorRef.noSender());

        // shut down the system
        system.terminate();
    }
}
```

Output:

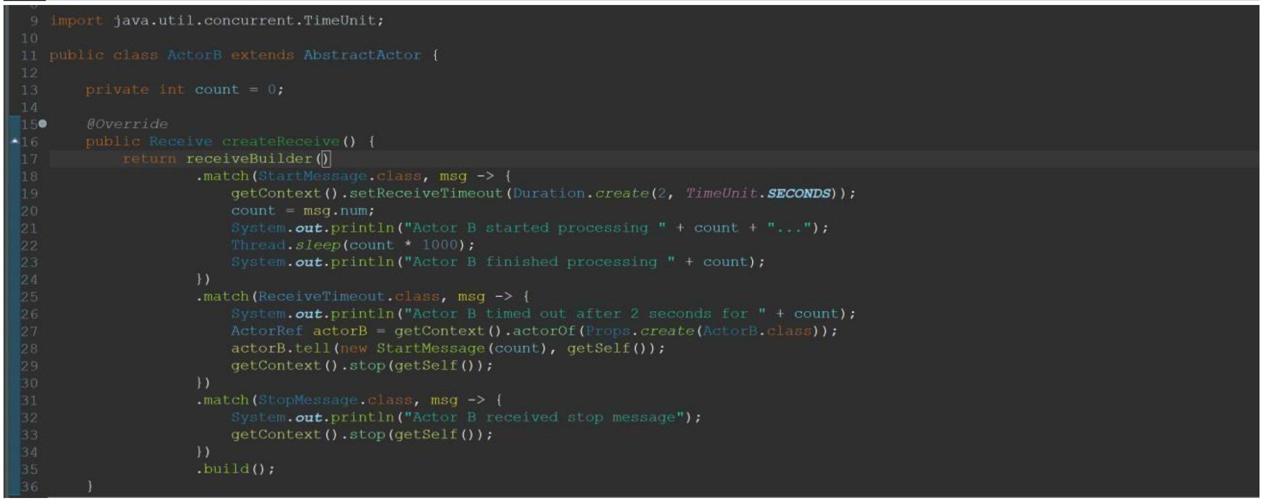


```
<terminated> Main (4) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe  (Mar 24, 2023, 4:27:04 PM – 4:27:07 PM) [pid: 25608]
Counter value is: 8 - Received message from Actor[akka://MySystem/user/actorA_5#991259908]
Counter value is: 9 - Received message from Actor[akka://MySystem/user/actorA_10#-563159252]
Counter value is: 10 - Received message from Actor[akka://MySystem/user/actorA_9#-1895822286]
Counter value is: 11 - Received message from Actor[akka://MySystem/user/actorA_18#-206593253]
Counter value is: 12 - Received message from Actor[akka://MySystem/user/actorA_4#705724275]
Counter value is: 13 - Received message from Actor[akka://MySystem/user/actorA_2#449187714]
Counter value is: 14 - Received message from Actor[akka://MySystem/user/actorA_8#661627409]
Counter value is: 15 - Received message from Actor[akka://MySystem/user/actorA_12#823806078]
Counter value is: 16 - Received message from Actor[akka://MySystem/user/actorA_13#-1387861363]
Counter value is: 17 - Received message from Actor[akka://MySystem/user/actorA_15#-642776004]
Counter value is: 18 - Received message from Actor[akka://MySystem/user/actorA_16#-486717131]
Counter value is: 19 - Received message from Actor[akka://MySystem/user/actorA_17#499930850]
Counter value is: 20 - Received message from Actor[akka://MySystem/user/actorA_19#82504906]
```

4. Create 2 Akka Actor classes "ActorA" and "ActorB" to demonstrate the Akka API setReceiveTimeout().ActorA will generate a random integer number from 1 to 5 in a loop for 100 times, and send this integer as a message to ActorB, which would then call Thread.sleep() that many seconds. Set the receive timeout to 2 seconds, and when the timeout triggers, send ActorB a stop() message, and then create a new instance of ActorB to process the next number.

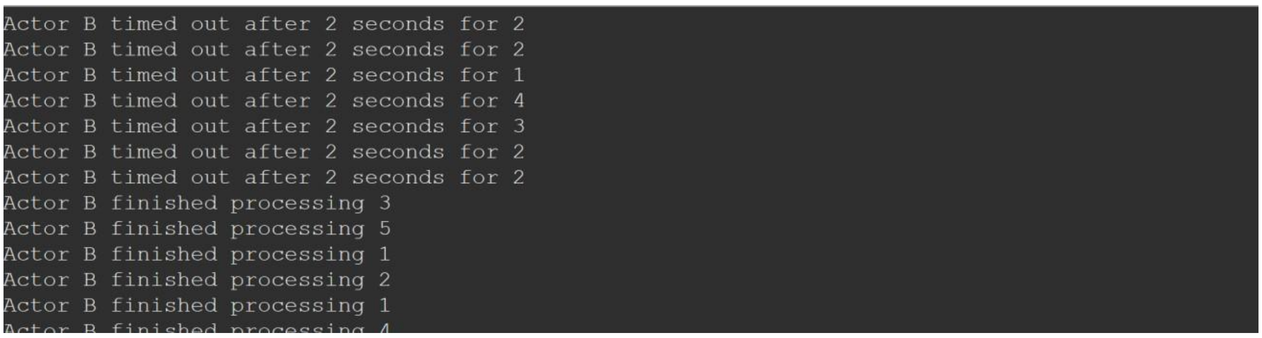


```java
package com.example;

import akka.actor.AbstractActor;
import akka.actor.ActorRef;
import akka.actor.Props;
import scala.concurrent.duration.Duration;

import java.util.Random;
import java.util.concurrent.TimeUnit;

public class ActorA extends AbstractActor {

    private Random random = new Random();

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .matchEquals("START", msg -> {
                    for (int i = 0; i < 100; i++) {
                        int num = random.nextInt(5) + 1;
                        ActorRef actorB = getContext().actorOf(Props.create(ActorB.class));
                        actorB.tell(new StartMessage(num), getSelf());
                    }
                })
                .build();
    }
}

```



```java
import java.util.concurrent.TimeUnit;

public class ActorB extends AbstractActor {

    private int count = 0;

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(StartMessage.class, msg -> {
                    getContext().setReceiveTimeout(Duration.create(2, TimeUnit.SECONDS));
                    count = msg.num;
                    System.out.println("Actor B started processing " + count + "...");
                    Thread.sleep(count * 1000);
                    System.out.println("Actor B finished processing " + count);
                })
                .match(ReceiveTimeout.class, msg -> {
                    System.out.println("Actor B timed out after 2 seconds for " + count);
                    ActorRef actorB = getContext().actorOf(Props.create(ActorB.class));
                    actorB.tell(new StartMessage(count), getSelf());
                    getContext().stop(getSelf());
                })
                .match(StopMessage.class, msg -> {
                    System.out.println("Actor B received stop message");
                    getContext().stop(getSelf());
                })
                .build();
    }
```

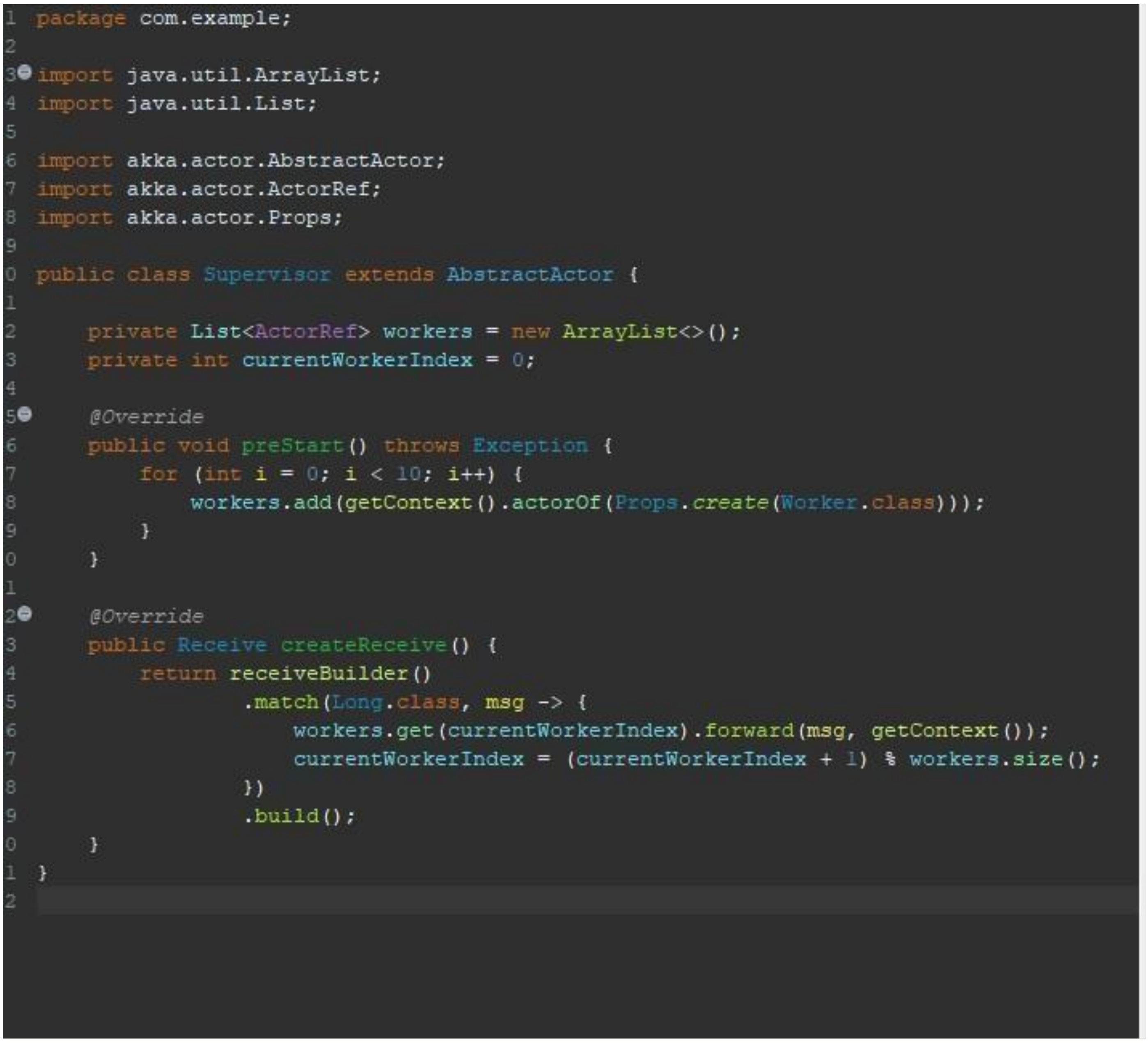
Output:


```
Actor B timed out after 2 seconds for 2
Actor B timed out after 2 seconds for 2
Actor B timed out after 2 seconds for 1
Actor B timed out after 2 seconds for 4
Actor B timed out after 2 seconds for 3
Actor B timed out after 2 seconds for 2
Actor B timed out after 2 seconds for 2
Actor B finished processing 3
Actor B finished processing 5
Actor B finished processing 1
Actor B finished processing 2
Actor B finished processing 1
Actor B finished processing 4
```

Assessed Task

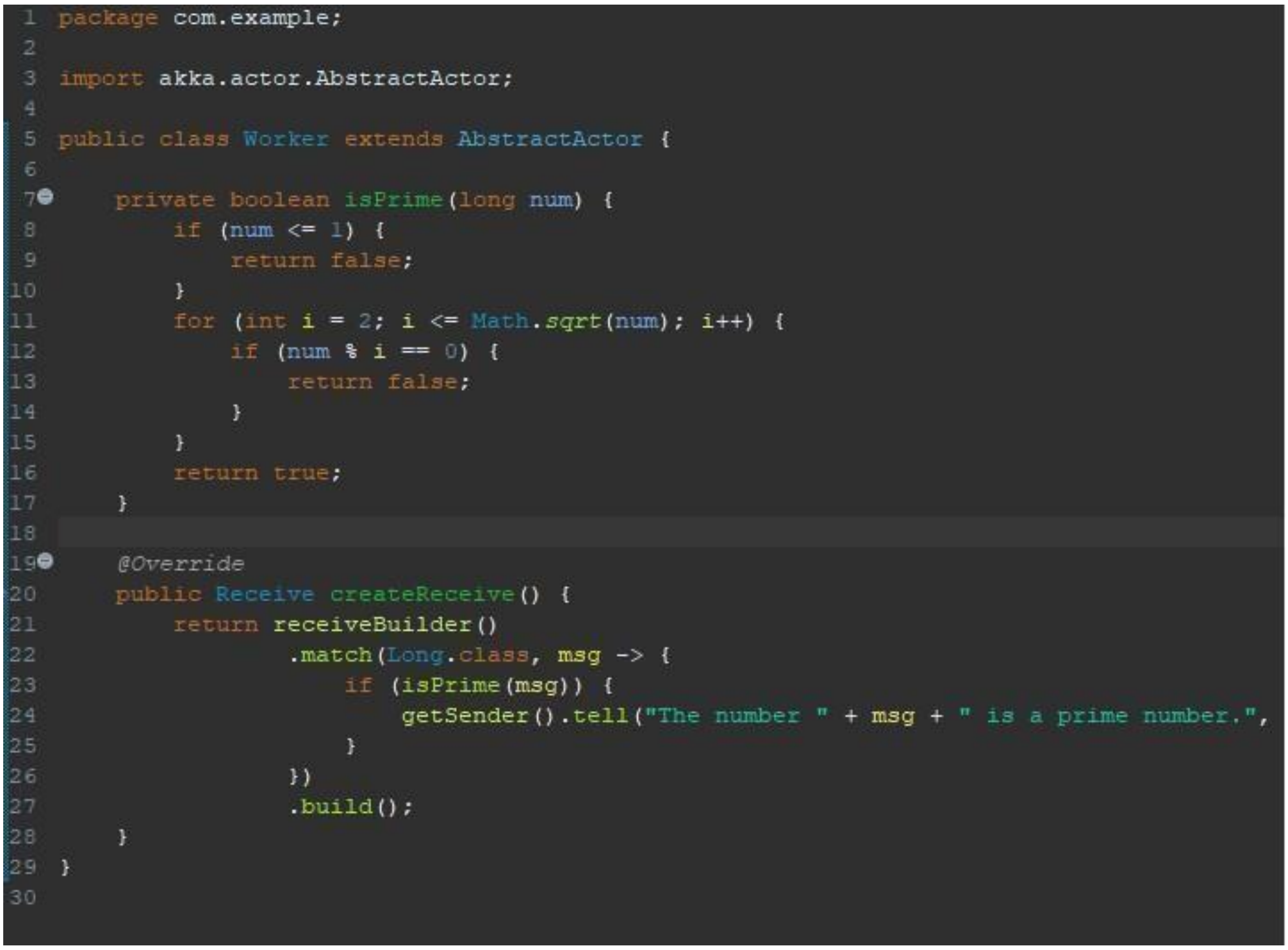
1. Create 3 Akka Actor classes called "Producer", "Supervisor" and "Worker". The "Producer" will generate 1000 random long integer numbers between 10000 and 100000. The "Producer" will send each number as a message to "Supervisor". At start-up, the Supervisor will create 10 "Worker" Actors. When the "Supervisor" receives a number from the "Producer", it will use the API forward() to forward that message to one of the "Worker" actors, in a round-robin fashion.

   Here, supervisor actor is creating 10 workers as the initial requirement of the assignment. Where it creates 10 workers as intended and uses



```java
package com.example;

import java.util.ArrayList;
import java.util.List;

import akka.actor.AbstractActor;
import akka.actor.ActorRef;
import akka.actor.Props;

public class Supervisor extends AbstractActor {

    private List<ActorRef> workers = new ArrayList<>();
    private int currentWorkerIndex = 0;

    @Override
    public void preStart() throws Exception {
        for (int i = 0; i < 10; i++) {
            workers.add(getContext().actorOf(Props.create(Worker.class)));
        }
    }

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(Long.class, msg -> {
                    workers.get(currentWorkerIndex).forward(msg, getContext());
                    currentWorkerIndex = (currentWorkerIndex + 1) % workers.size();
                })
                .build();
    }
}
```

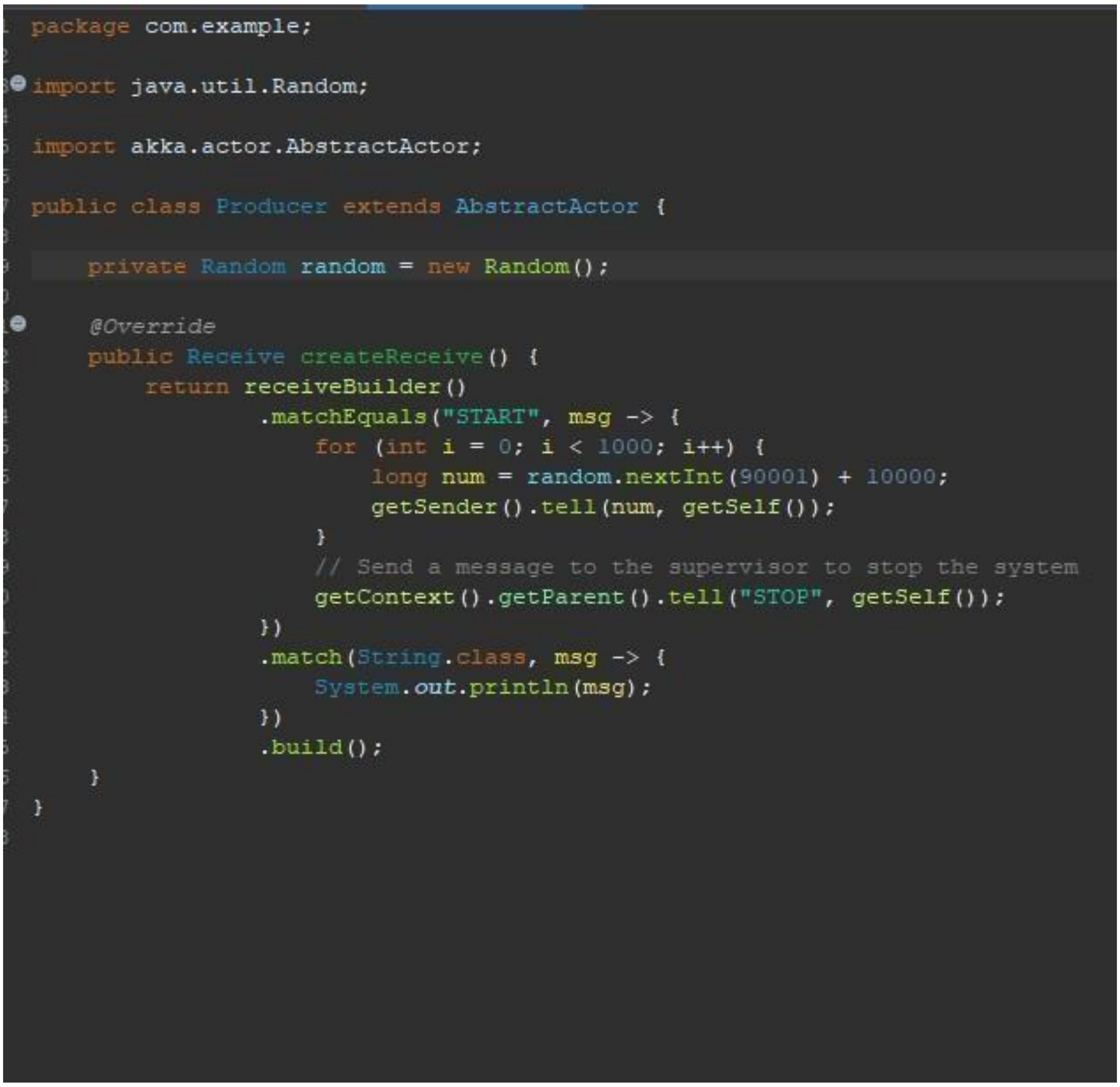
The "Worker" actor will determine if the number in the message is a prime number. If it is a prime number, it will then send a string/text message to the "Producer", saying that "The number XXX is a prime number." And the Producer will print out the message on the standard output.   Here, workers are checking if the number is either prime or not which is produced by producer.



```java
package com.example;

import akka.actor.AbstractActor;

public class Worker extends AbstractActor {

    private boolean isPrime(long num) {
        if (num <= 1) {
            return false;
        }
        for (int i = 2; i <= Math.sqrt(num); i++) {
            if (num % i == 0) {
                return false;
            }
        }
        return true;
    }

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(Long.class, msg -> {
                    if (isPrime(msg)) {
                        getSender().tell("The number " + msg + " is a prime number.",
                    }
                })
                .build();
    }
}
```

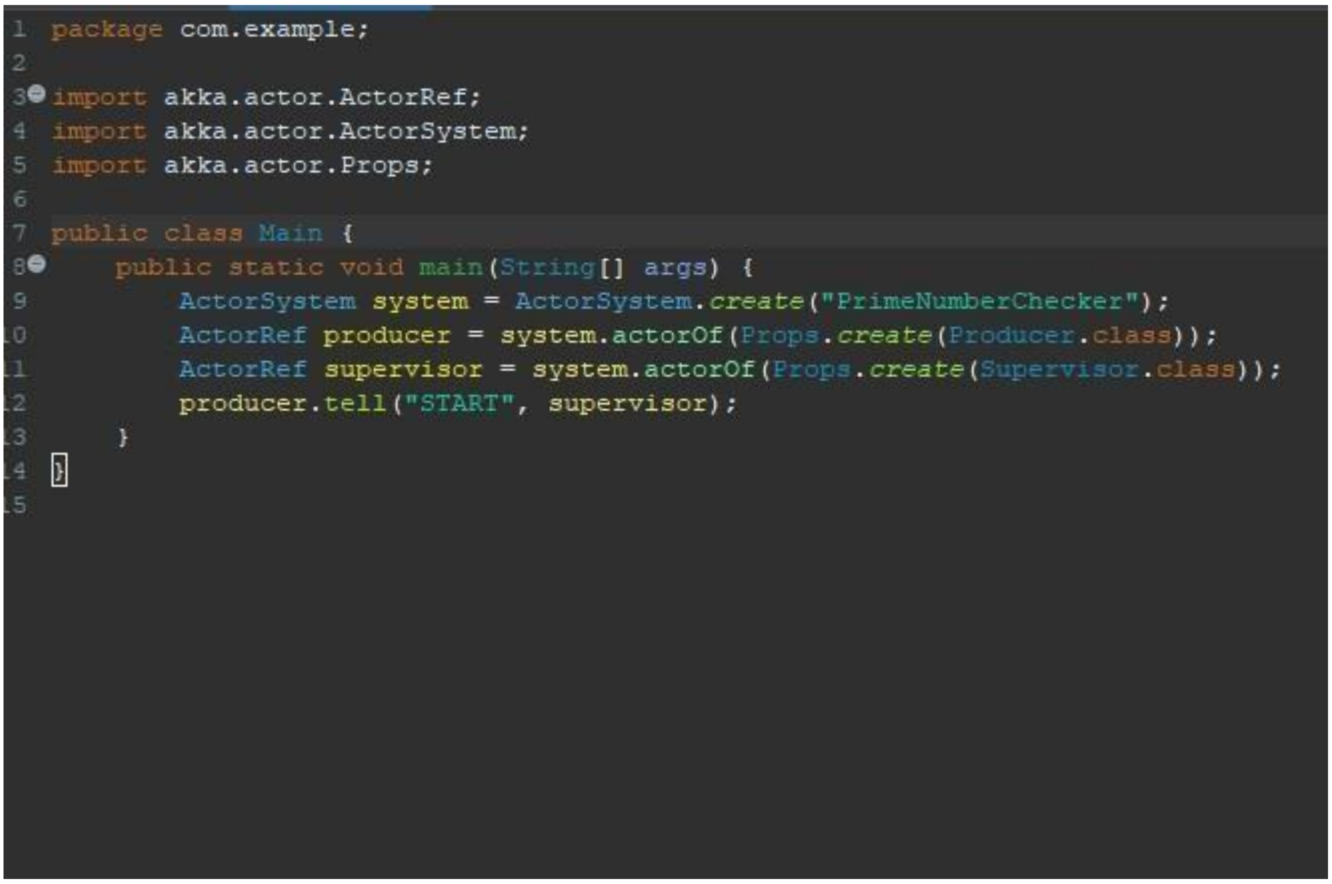
When the 1000 numbers have been produced and checked, the "Producer" actor will terminate the Actor system.

Here Producer produce prime numbers upto 1000.



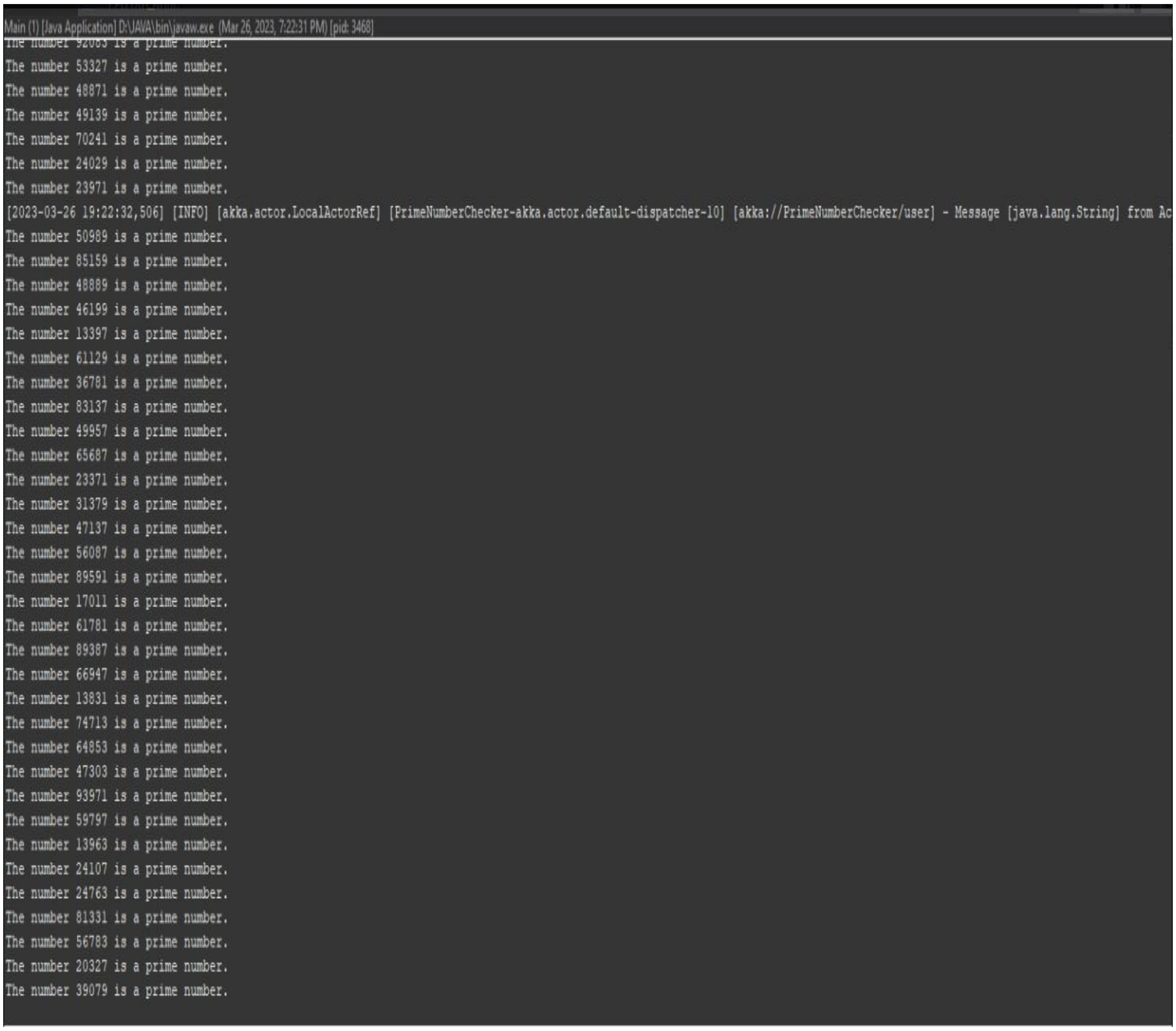
```java
package com.example;

import java.util.Random;

import akka.actor.AbstractActor;

public class Producer extends AbstractActor {

    private Random random = new Random();

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .matchEquals("START", msg -> {
                    for (int i = 0; i < 1000; i++) {
                        long num = random.nextInt(90001) + 10000;
                        getSender().tell(num, getSelf());
                    }
                    // Send a message to the supervisor to stop the system
                    getContext().getParent().tell("STOP", getSelf());
                })
                .match(String.class, msg -> {
                    System.out.println(msg);
                })
                .build();
    }
}
```

Main file to run the code:



```java
package com.example;

import akka.actor.ActorRef;
import akka.actor.ActorSystem;
import akka.actor.Props;

public class Main {
    public static void main(String[] args) {
        ActorSystem system = ActorSystem.create("PrimeNumberChecker");
        ActorRef producer = system.actorOf(Props.create(Producer.class));
        ActorRef supervisor = system.actorOf(Props.create(Supervisor.class));
        producer.tell("START", supervisor);
    }
}

```

Output:



```
Main (1) [Java Application] D:\JAVA\bin\javaw.exe (Mar 26, 2023, 7:22:31 PM) [pid: 3468]
The number 92083 is a prime number.
The number 53327 is a prime number.
The number 48871 is a prime number.
The number 49139 is a prime number.
The number 70241 is a prime number.
The number 24029 is a prime number.
The number 23971 is a prime number.
[2023-03-26 19:22:32,506] [INFO] [akka.actor.LocalActorRef] [PrimeNumberChecker-akka.actor.default-dispatcher-10] [akka://PrimeNumberChecker/user] - Message [java.lang.String] from Ac
The number 50989 is a prime number.
The number 85159 is a prime number.
The number 48889 is a prime number.
The number 46199 is a prime number.
The number 13397 is a prime number.
The number 61129 is a prime number.
The number 36781 is a prime number.
The number 83137 is a prime number.
The number 49957 is a prime number.
The number 65687 is a prime number.
The number 23371 is a prime number.
The number 31379 is a prime number.
The number 47137 is a prime number.
The number 56087 is a prime number.
The number 89591 is a prime number.
The number 17011 is a prime number.
The number 61781 is a prime number.
The number 89387 is a prime number.
The number 66947 is a prime number.
The number 13831 is a prime number.
The number 74713 is a prime number.
The number 64853 is a prime number.
The number 47303 is a prime number.
The number 93971 is a prime number.
The number 59797 is a prime number.
The number 13963 is a prime number.
The number 24107 is a prime number.
The number 24763 is a prime number.
The number 81331 is a prime number.
The number 56783 is a prime number.
The number 20327 is a prime number.
The number 39079 is a prime number.
```