

# Sesión 4: Espera Activa (*busy waiting*)

Concurrencia

---

Ángel Herranz

2019-2020

Universidad Politécnica de Madrid

# Concurrencia

 Simultaneidad

+

Sincronización +  Comunicación<sup>1</sup>

---

<sup>1</sup>Sólo con memoria compartida.

# Concurrencia

 Simultaneidad

+

Sincronización +  Comunicación<sup>1</sup>

---

<sup>1</sup>Sólo con memoria compartida.

## Concepto: *condición de carrera*

Resultados *indeseados* por *interacción*  
de dos o más procesos que *leen y*  
*modifican* datos *compartidos*  
  
=  
  
*condición de carrera*



## Concepto: *sección crítica*

Porción de código que puede dar lugar  
a una condición de carrera

=

*sección crítica*



# Secciones críticas

```
public static int x = 0;
```

```
public class Inc
    extends Thread {
    public void run() {
        x = x + 1;
    }
}
```

```
public class Dec
    extends Thread {
    public void run() {
        x = x - 1;
    }
}
```



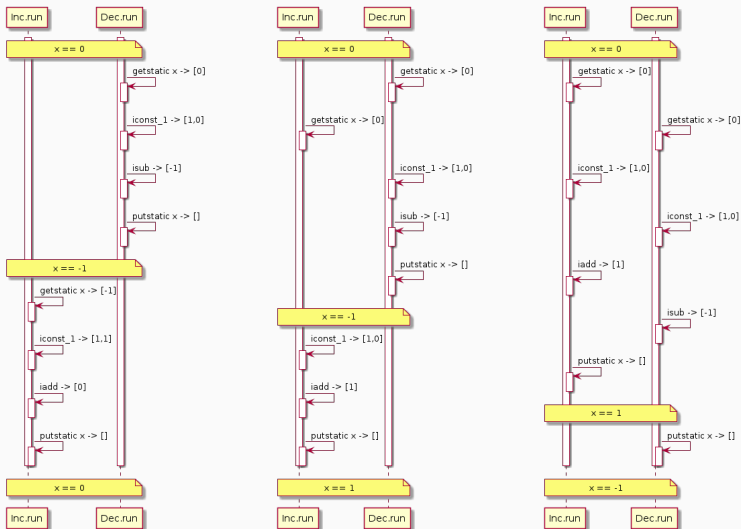
# Secciones críticas ¿Por qué?

```
public static int x = 0;
```

```
public class Inc
    extends Thread {
    public void run() {
        x = x + 1;
    }
}
```

```
public class Dec
    extends Thread {
    public void run() {
        x = x - 1;
    }
}
```

# Semántica: *entrelazados de acciones atómicas*





## Solución: Exclusión mutua

Mientras un proceso está ejecutando  
la sección crítica, los otros procesos  
deben esperar

=

Exclusión mutua

## Solución: Exclusión mutua

Mientras un proceso está ejecutando  
la sección crítica, los otros procesos

deben esperar

=

Exclusión mutua

¡Sincronización!

## Aviso: abstracción, por favor

- A lo largo de varias sesiones vamos a usar `x = x + 1;` como *ejemplo* de sección crítica
- *Abstraed*, por favor: podeis imaginar otros ejemplos más *contundentes* de código *normal* que en concurrencia son secciones críticas

```
// Construcción de un "singleton"  
if (singleton == null)  
    singleton = new Database();  
return singleton;
```

## Aviso: abstracción, por favor

- A lo largo de varias sesiones vamos a usar `x = x + 1;` como *ejemplo* de sección crítica
- *Abstraed*, por favor: podeis imaginar otros ejemplos más *contundentes* de código *normal* que en concurrencia son secciones críticas

```
// Insertar si no está
if (!map.contains(key)) {
    map.put(key, value);
}
```

## Aviso: abstracción, por favor

- A lo largo de varias sesiones vamos a usar `x = x + 1;` como *ejemplo* de sección crítica
- *Abstraed*, por favor: podeis imaginar otros ejemplos más *contundentes* de código *normal* que en concurrencia son secciones críticas

```
// Actualizar el saldo de una cuenta  
cuenta = repo.get(id);  
cuenta.saldo += transferencia.value;  
repo.save(cuenta);
```

# ¿Cómo esperar?

- Sin ayuda del lenguaje o bibliotecas.
- Programando con *nuestras propias manos*
- Teniendo en cuenta todos los entrelazados.



¿Cómo esperar hasta que se cumpla *C*?

# ¿Cómo esperar?

- Sin ayuda del lenguaje o bibliotecas.
- Programando con *nuestras propias manos*
- Teniendo en cuenta *todos los entrelazados*.



¿Cómo esperar hasta que se cumpla *C*?

```
while (!C) {  
    // no hacer nada  
}  
// ¡Aquí se cumple C!
```

# ¿Cómo esperar?

- Sin ayuda del lenguaje o bibliotecas.
- Programando con *nuestras propias manos*
- Teniendo en cuenta *todos los entrelazados*.



¿Cómo esperar hasta que se cumpla *C*?

```
while (!C) {  
    // no hacer nada  
}  
// ¡Aquí se cumple C!
```

- Concepto: sincronización por espera activa



# Primer intento

```
public static volatile int x = 0;
```

```
x = x + 1;
```

```
x = x - 1;
```

# Primer intento

```
public static volatile int x = 0;  
public static volatile boolean enSC = false;
```

```
x = x + 1;
```

```
x = x - 1;
```



enSC  $\Leftrightarrow$  “un proceso está en sección crítica”

# Primer intento

```
public static volatile int x = 0;  
public static volatile boolean enSC = false;
```

```
enSC = true;  
x = x + 1;  
enSC = false;
```

```
enSC = true;  
x = x - 1;  
enSC = false;
```



enSC  $\Leftrightarrow$  “un proceso está en sección crítica”

# Primer intento

```
public static volatile int x = 0;  
public static volatile boolean enSC = false;
```

```
    while (enSC) {}  
    enSC = true;  
    x = x + 1;  
    enSC = false;
```



enSC  $\Leftrightarrow$  “un proceso está en sección crítica”

# Primer intento

```
public static volatile int x = 0;  
public static volatile boolean enSC = false;
```

```
    while (enSC) {}  
    enSC = true;  
    x = x + 1;  
    enSC = false;
```



enSC  $\Leftrightarrow$  “un proceso está en sección crítica”



¿Qué va a pasar?

# CC\_02\_Carrera.java con dos procesos



```
1 public class EsperaActiva {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean enSC = false;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (enSC) {}
10                enSC = true;
11                x = x + 1;
12                enSC = false;
13            }
14        }
15    }
```

 variables compartidas,

```
17 static class Dec extends Thread {
18     public void run() {
19         for (int i = 0; i < N_OPS; i++) {
20             while (enSC) {}
21             enSC = true;
22             x = x - 1;
23             enSC = false;
24         }
25     }
26 }
27
28 public static void main(String[] args)
29     throws InterruptedException {
30     Thread i = new Inc();
31     Thread d = new Dec();
32     i.start(); d.start();
33     i.join(); d.join();
34     System.out.println(x);
35 }
36 }
```

# CC\_02\_Carrera.java con dos procesos



```
1 public class EsperaActiva {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean enSC = false;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (enSC) {}
10                enSC = true;
11                x = x + 1;
12                enSC = false;
13            }
14        }
15    }
```

 variables compartidas,  secciones críticas,

```
17 static class Dec extends Thread {
18     public void run() {
19         for (int i = 0; i < N_OPS; i++) {
20             while (enSC) {}
21             enSC = true;
22             x = x - 1;
23             enSC = false;
24         }
25     }
26 }
27
28 public static void main(String[] args)
29     throws InterruptedException {
30     Thread i = new Inc();
31     Thread d = new Dec();
32     i.start(); d.start();
33     i.join(); d.join();
34     System.out.println(x);
35 }
36 }
```

# CC\_02\_Carrera.java con dos procesos

```
1 public class EsperaActiva {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean enSC = false;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (enSC) {}
10                enSC = true;
11                x = x + 1;
12                enSC = false;
13            }
14        }
15    }
```




 variables compartidas,  secciones críticas,

```
17 static class Dec extends Thread {
18     public void run() {
19         for (int i = 0; i < N_OPS; i++) {
20             while (enSC) {}
21             enSC = true;
22             x = x - 1;
23             enSC = false;
24         }
25     }
26 }
27
28 public static void main(String[] args)
29     throws InterruptedException {
30     Thread i = new Inc();
31     Thread d = new Dec();
32     i.start(); d.start();
33     i.join(); d.join();
34     System.out.println(x);
35 }
36 }
```



# CC\_02\_Carrera.java con dos procesos





```
1 public class EsperaActiva {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean enSC = false;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (enSC) {}
10                enSC = true;
11                x = x + 1;
12                enSC = false;
13            }
14        }
15    }
```

 variables compartidas,  secciones críticas,  protocolo de entrada a la sección crítica,

```
17 static class Dec extends Thread {
18     public void run() {
19         for (int i = 0; i < N_OPS; i++) {
20             while (enSC) {}
21             enSC = true;
22             x = x - 1;
23             enSC = false;
24         }
25     }
26 }
27
28 public static void main(String[] args)
29     throws InterruptedException {
30     Thread i = new Inc();
31     Thread d = new Dec();
32     i.start(); d.start();
33     i.join(); d.join();
34     System.out.println(x);
35 }
36 }
```

# CC\_02\_Carrera.java con dos procesos

```
1 public class EsperaActiva {
2     static final int N_OPS = 1000;
3     static volatile int x = 0;
4     static volatile boolean enSC = false;
5
6     static class Inc extends Thread {
7         public void run() {
8             for (int i = 0; i < N_OPS; i++) {
9                 while (enSC) {}
10                enSC = true;
11                x = x + 1;
12                enSC = false;
13            }
14        }
15    }
```

 variables compartidas,  secciones críticas,  protocolo de entrada a la sección crítica,  protocolo de salida

```
17     static class Dec extends Thread {
18         public void run() {
19             for (int i = 0; i < N_OPS; i++) {
20                 while (enSC) {}
21                 enSC = true;
22                 x = x - 1;
23                 enSC = false;
24             }
25         }
26     }
27
28     public static void main(String[] args)
29         throws InterruptedException {
30         Thread i = new Inc();
31         Thread d = new Dec();
32         i.start(); d.start();
33         i.join(); d.join();
34         System.out.println(x);
35     }
36 }
```