ESCOLA TÈCNICA SUPERIOR D'ENGINYERIA DE TELECOMUNICACIÓ
DEPARTAMENT D'ENGINYERIA TELEMÀTICA
APLICACIONS I SERVEIS TELEMÀTICS
Nivell de Transport

Interfícies del Nivell de Xarxa i Nivell de Transport

En les activitats es demanen diferents implementacions de la classe `TSocket`.

```
public interface Xarxa {
    public void enviar(Object objecte);
    public Object rebre();
    public void close();
}
```

```
public abstract class TSocket{
    protected Xarxa xarxa;
    // En les subclasses de TSocket s'utilitzara
    // xarxa.enviar(objecte a enviar)
    // per enviar dades a la xarxa

    protected ReentrantLock mon;
    protected Condition appCV;

    public TSocket(Xarxa x) {
        xarxa = x;
        mon = new ReentrantLock();
        appCV = mon.newCondition();
        new Thread(new PartReceptora()).start();
    }


    public abstract void processarMissatge(Segment seg);

    class PartReceptora implements Runnable {
        public void run() {
            while (true) {
                Object rebut = null;
                rebut = (Segment)xarxa.rebre();
                processarMissatge(rebut);
            }
        }
    }

    public Object rebre(){
        throw new RuntimeException("Implementar en les subclasses");
    }

    public void enviar(Object miss){
        throw new RuntimeException("Implementar en les subclasses");
    }
}
```

```
public interface Comms {
    public final static int MIDA_CUA_RECEPCIO = 50;
    public final static int MAX_INT_EMISSOR = 200;
    public final static int DADES = 100;

    public final static int ACK = 200;
    public final static int CONG = 300;

    public static final int RTO = 500; //timeout retransmissio en millisegons
    public static final int RTT = 600; //Round Trip Time
    public static final int RTT_INF = 450; //Round Trip Time inf
    public static final int RTT_SUP = 550; //Round Trip Time sup

    public static final int FINESTRA_CONGESTIO = 4;
}
```

```java
public class Emissor implements Runnable {

    protected TSocket socket;

    public Emissor(TSocket s) {
        socket = s;
    }

    public void run() {
        int num = 0;
        while (num < Comms.MAX_INT_EMISSOR) {
            adormir(500);
            socket.enviar(num);
            System.out.println("num enviat "+num);
            num = num + 1;
        }
    }
}

public class Receptor implements Runnable {

    protected TSocket socket;

    public Receptor(TSocket s) {
        socket = s;
    }

    public void run() {
        while (true) {
            adormir(1000);
            Object rebut = socket.rebre();
            System.out.println("Receptor: " + rebut);
        }
    }
}
```

```java
public class Segment implements Serializable{
    protected int tipus;
    protected int numeroSequencia;
    protected int finestra;
    protected Object dades;

    public Segment(int tipus, int numeroSequencia, Object dades){
        this.tipus = tipus;
        this.numeroSequencia = numeroSequencia;
        this.dades = dades;
        this.finestra = -1;
    }

    public Segment(int tipus, int numeroSequencia, int finestra, Object dades){
        this.tipus = tipus;
        this.numeroSequencia = numeroSequencia;
        this.dades = dades;
        this.finestra = finestra;
    }

    public int getNumSeq(){ return numeroSequencia; }

    public Object getDades(){ return dades; }

    public int getFinestra(){ return finestra; }

    public int getTipus(){ return tipus; }

    public String toString(){
        return new String("tipus: "+ tipus +" numSeq: " + numeroSequencia +
                          " valor: " + dades + " finestra: " + finestra);
    }
}
```

Control Flux. Sense Pèrdues.

Es vol simular el control de flux del protocol TCP. Per simplicitat en cada segment enviat les dades es limiten a un únic caràcter (o enter). La quantitat de segments que es poden enviar en un instant determinat, està indicat per la variable `finestraRecepcio` de la classe `ProtocolEnviarControlFlux`.
Aspectes a considerar:

- Per simplicitat **totes** les variables enteres poden ser incrementades sense límit, és a dir, se suposa que no hi ha overflow.
- L'atribut `numeroSequencia` de la classe `Segment`, indica el número de seqüència si el segment és de dades, i el número del segment que reconeix si és un segment d'ACK.
- En els segments d'ACK el valor de les dades es `null`.
- La cua circular en la classe `ProtocolRebreControlFlux` guarda els caràcters (o enters) rebuts. Els mètodes de que disposa són els habituals.
- El mètode `enviar(Objecte c)` és bloquejant si enviar dades suposa superar la `finestraRecepcio`.

```
public class TSocketEnviarControlFlux extends TSocket {

    protected int seguentEnviar, seguentASerReconegut, finestraRecepcio;

    public TSocketEnviarControlFlux(Xarxa x) {
        super(x);
        finestraRecepcio = Comms.MIDA_CUA_RECEPCIO;
    }

    @Override
    public void enviar(Object c) {
        try {
            mon.lock();
            //Per enviar segments: xarxa.enviar(seg);

            throw new RuntimeException("Part a completar");



        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




//              System.out.println("ACK rebut -> " + "finestra recepcio: "
//                      + finestraRecepcio);

        } finally {
            mon.unlock();
        }
    }
}


public class TSocketRebreControlFlux extends TSocket {

    protected int seguentARebre;
    protected CircularQueue cuaRecepcio;

    public TSocketRebreControlFlux(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue(Comms.MIDA_CUA_RECEPCIO);
    }
```

```
    @Override
    public Object rebre() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();
            return null;
        } finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            mon.unlock();
        }
    }
}
```

Control Flux. Finestra Zero. Sense Pèrdues.

Ara es demana una implementació complerta de l'exercici anterior incloent el cas de "*finestra zero*".

```
public class TSocketEnviarControlFluxFinestra0 extends TSocket {

    protected int seguentEnviar, seguentReconeixer, finestraRecepcio;
    protected Segment segmentSondejar;

    protected Timer timer;
    protected Timer.Task tascaTimer;

    public TSocketEnviarControlFluxFinestra0(Xarxa x) {
        super(x);
        finestraRecepcio = Comms.MIDA_CUA_RECEPCIO;
        timer = new Timer();
    }

    @Override
    public void enviar(Object c) {
        try {
        mon.lock();

            while (((seguentEnviar - seguentReconeixer) >= finestraRecepcio)
                    && finestraRecepcio!=0) {
                appCV.await();
            }

            // si la finestra es zero, el segmentSondejaro que envio anira sent
            // descartat en recepcio, m'he d'esperar fins que sigui
            // reconegut i aixo passara quan :
            // seguentEnviar == seguentReconeixer
            //
            // sino
            //
            // Envio el segment, no cal que el guardi perque no hi ha perdues

            if(finestraRecepcio==0){
              throw new RuntimeException("Part a completar");









            }else{
              throw new RuntimeException("Part a completar");









            }
        } catch (Exception ex) { System.out.println(ex); }
        finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } finally {
            mon.unlock();
        }
    }

    /**
     * Timer expira
     */
    protected void timeout() {
        mon.lock();
        try {
```

```
            assert (finestraRecepcio == 0 && segmentSondejar != null);
            throw new RuntimeException("Part a completar");



//            System.out.println("reenviat : " + segmentSondejar);
        } finally {
            mon.unlock();
        }
    }

    protected void startRTO() {
        if (tascaTimer != null) {
            tascaTimer.cancel();
        }
        tascaTimer = timer.startAfter(
                new Runnable() {
                    @Override
                    public void run() {
                        timeout();
                    }
                },
                Comms.RTO, TimeUnit.MILLISECONDS);
    }

    protected void stopRTO() {
        if (tascaTimer != null) {
            tascaTimer.cancel();
        }
        tascaTimer = null;
    }

}


public class TSocketRebreControlFluxFinestra0 extends TSocket {

    protected int seguentARebre;
    protected CircularQueue cuaRecepcio;

    public TSocketRebreControlFluxFinestra0(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue(Comms.MIDA_CUA_RECEPCIO);
    }

    public Object rebre() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();
            return null;
        } finally {
            mon.unlock();
        }
    }

    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            mon.unlock();
        }
    }
}
```

Control Flux. Finestra Zero. Stop And Wait. Sense Pèrdues.

Ara es demana una implementació del protocol Stop and Wait incloent el cas de "*finestra zero*", però sense pèrdues de segments. Per això s'introdueix en l'emissor una variable `finestraPermesa` amb valors possibles $\{0, 1\}$.

```
public class TSocketEnviarFluxFinestra0StopandWait extends TSocket {

    protected int seguentEnviar, seguentASerReconegut;
    protected int finestraRecepcio;
    protected int finestraPermesa; // Valors possibles {0,1}
    protected Segment segmentTimeout;

    protected Timer timer;
    protected Timer.Task tascaTimer;

    public TSocketEnviarFluxFinestra0StopandWait(Xarxa x) {
        super(x);
        finestraRecepcio = Comms.MIDA_CUA_RECEPCIO;
        finestraPermesa = 1;
        timer = new Timer();
    }

    public void enviar(Object c) {
        try {
            mon.lock();
            throw new RuntimeException("Part a completar");









        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            mon.unlock();
        }
    }

    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");







        } finally {
            mon.unlock();
        }
    }

    /**
     * Timer expira
     */
    protected void timeout() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");



//            System.out.println("reenviat : " + segmentTimeout);
        } finally {
            mon.unlock();
        }
    }

    protected void startRTO() {
        if (tascaTimer != null) {
            tascaTimer.cancel();
        }
```

```
        tascaTimer = timer.startAfter(
                new Runnable() {
                    @Override
                    public void run() {
                        timeout();
                    }
                },
                Comms.RTO, TimeUnit.MILLISECONDS);
    }

    protected void stopRTO() {
        if (tascaTimer != null) {
            tascaTimer.cancel();
        }
        tascaTimer = null;
    }
}


public class TSocketRebreFluxFinestra0StopandWait extends TSocket {

    protected int seguentARebre;
    protected CircularQueue cuaRecepcio;

    public TSocketRebreFluxFinestra0StopandWait(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue(Comms.MIDA_CUA_RECEPCIO);
    }

    @Override
    public Object rebre() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");









        } catch (Exception e) {
            e.printStackTrace();
            return null;
        } finally {
            mon.unlock();
        }
    }

    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");

















        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            mon.unlock();
        }
    }
}
```

Control Flux. Finestra Zero. Stop And Wait. Amb Pèrdues.

Ara es demana una implementació complerta de l'exercici anterior, és a dir, amb pèrdues de segments.

```
public class TSocketEnviarControlFluxStopAndWaitARQ extends TSocket {

    protected int seguentEnviar, seguentASerReconegut;
    protected int finestraRecepcio, finestraPermesa;
    protected Segment segTimeout;

    protected Timer timer;
    protected Timer.Task tascaTimer;

    public TSocketEnviarControlFluxStopAndWaitARQ(Xarxa x) {
        super(x);
        finestraRecepcio = Comms.MIDA_CUA_RECEPCIO;
        finestraPermesa = 1;
        timer = new Timer();
    }

    public void enviar(Object c) {
        try {
            mon.lock();
            throw new RuntimeException("Part a completar");









        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            mon.unlock();
        }
    }

    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            // Simula perdua de d'ack
            double llindar = Math.random();
            if (llindar > 0.8) {
                System.out.println("ack perdut : " + segment);
                return;
            }

            throw new RuntimeException("Part a completar");
















        } finally {
            mon.unlock();
        }
    }

    /**
     * Timer expira
     */
    protected void timeout() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




//            System.out.println("reenviat : " + segTimeout);
        } finally {
            mon.unlock();
        }
```

```
    }

    protected void startRTO() {
        if (tascaTimer != null) {
            tascaTimer.cancel();
        }
        tascaTimer = timer.startAfter(
                new Runnable() {
                    @Override
                    public void run() {
                        timeout();
                    }
                },
                Comms.RTO, TimeUnit.MILLISECONDS);
    }

    protected void stopRTO() {
        if (tascaTimer != null) {
            tascaTimer.cancel();
        }
        tascaTimer = null;
    }

}


public class TSocketRebreControlFluxStopAndWaitARQ extends TSocket {

    protected int seguentARebre;
    protected CircularQueue cuaRecepcio;

    public TSocketRebreControlFluxStopAndWaitARQ(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue(Comms.MIDA_CUA_RECEPCIO);
    }

    public Object rebre() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();return null;
        } finally {
            mon.unlock();
        }
    }

    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            // Simula perdua de segment
            double llindar = Math.random();
            if (llindar > 0.8) {
                System.out.println("segment dades perdut : " + segment);
                return;
            }

            throw new RuntimeException("Part a completar");







        } catch (Exception e) { e.printStackTrace();
        } finally {
            mon.unlock();
        }
    }
}
```

Control Congestió. Slow Start. Es vol simular la fase d'slow start del control de congestió del protocol TCP. La quantitat de segments que es poden enviar en un instant determinat, està indicat per la variable `finestraPermesa` de la classe `TSocketEnviarSlowStart`.
Aspectes a considerar:

- La variable `finestraCongestio` augmenta com en l'slow start de TCP, és a dir, per cada `ACK` rebut, es pot enviar un enter de més.
- En aquest cas la variable `finestraCongestio` pot creixer sense límit, és a dir, no es considera cap llindar en el creixement exponencial.
- La classe `TSocketRebreSlowStart` guarda els segments en una cua de tipus `ArrayList`. D'aquesta manera en un segment ja es pot enviar reconeixement.
- El mètode `enviar()` és bloquejant si enviar un segment suposa superar la `finestraPermesa`.

```
public class TSocketEnviarControlFluxSlowStart extends TSocket {

    protected int seguentEnviar, seguentASerReconegut;
    protected int finestraRecepcio, finestraCongestio, finestraPermesa;

    public TSocketEnviarControlFluxSlowStart(Xarxa x) {
        super(x);
        finestraRecepcio = Comms.MIDA_CUA_RECEPCIO;
        finestraCongestio = 1;
        finestraPermesa = 1;
    }

    @Override
    public void enviar(Object c) {
        try {
            mon.lock();
            throw new RuntimeException("Part a completar");




        } catch (Exception ex) {
            ex.printStackTrace();
        } finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




//              System.out.println("ACK rebut -> " + "finestra permesa: "
//                      + finestraPermesa);
        } finally {
            mon.unlock();
        }
    }

}
```

```java
public class TSocketRebreControlFluxSlowStart extends TSocket {

    protected int NextToBeRcv;
    protected CircularQueue cuaRecepcio;

    public TSocketRebreControlFluxSlowStart(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue(Comms.MIDA_CUA_RECEPCIO);
    }

    @Override
    public Object rebre() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();
            return null;
        } finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Segment segment) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } catch (Exception e) {
            e.printStackTrace();
        } finally {
            mon.unlock();
        }
    }
}
```

TCP Vegas Style.

Es vol simular el control de congestió d'un protocol de transport. El mecanisme està inspirat en la versió TCP Vegas i considerarem una comunicació en un sol sentit. La part emissora del protocol s'implementarà en la classe `TSocketEnviar` i la part receptora en la classe `TSocketRebre`.

L'idea general és considerar un valor de referència de round trip time `(Comms.RTT)` fixe. En la classe TSocketEnviar es calcula el temps que passa des de que s'envia un segment fins que es rep el seu segment d'ACK corresponent. Es compara aquest temps amb `(Comms.RTT)` i s'incrementa o decrementa la variable `finestraCongestio` en una unitat segons correspongui.

Aspectes generals:

1. **La quantitat de segments que l'emissor ha injectat en la xarxa en un instant determinat no pot superar el valor de** `finestraCongestio`.
2. Per simplicitat se suposa que no hi ha pèrdues ni errors **ni reordenaments**.
3. Per simplicitat els segments no encapsulen arrays de bytes com a les pràctiques sino objectes.
4. En les classes `TSocketEnviar` i `TSocketRebre` es disposa d'un thread que crida al mètode `processReceivedSegment` quan arriba un segment des de la xarxa.
5. Per enviar i rebre segments s'utilitzen els mètodes `enviar` i `rebre` de `Xarxa`.

Aspectes a considerar en `TSocketEnviar` són:

1. Per saber l'instant actual de temps es disposa de la sentència `long System.currentTimeMillis()`.
2. Observar que en rebre un ACK, el temps d'enviament del segment que aquest ACK reconeix és el que hi ha a la posició 0 de l'`ArrayList tempsEnviament`. Per recuperar el temps cal fer un `remove` d'aquesta posició 0. Per guardar un element a l'`ArrayList` es disposa del mètode `add`.
3. El mètode `actualitzaFinestraCongestio` assigna a `finestraCongestio` el valor adient.
4. El mètode `enviar` es bloquejant si enviar un segment suposa superar la `finestraCongestio`.
5. En els segments de dades el tipus és `Comms.DADES`.

Aspectes a considerar en `TSocketRebre` són:

1. El mètode `rebre` es bloquejant si no hi ha dades per l'aplicació.
2. En els segments d'ACK el tipus és `Comms.ACK` i el parametre dades és `null`.

```
public class TSocketEnviarCongestioEstilVegas extends TSocket{
    protected int seguentEnviar, seguentASerReconegut,
                  finestraCongestio;
    protected HashMap<Integer, Long> tempsEnviament;


    public TSocketEnviarCongestioEstilVegas(Xarxa x) {
        super(x);
        tempsEnviament = new HashMap();
        finestraCongestio = 4;

    }
```

```
    public void enviar(Object c) {
        try {
        mon.lock();
        // Per simular diferents retards de segment fer servir:
        // SegmentDelayed segment = new SegmentDelayed(Comms.DADES, seguentEnviar, 0, c, random());
        // tempsEnviament.put(seguentEnviar, System.currentTimeMillis());

            throw new RuntimeException("Part a completar");




        } catch (Exception ex) { System.out.println(ex); }
        finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Segment segment) {
        mon.lock();
        try{
            throw new RuntimeException("Part a completar");



        }finally{
            mon.unlock();
        }
    }

    private void actualitzaFinestraCongestio(int numSeq){
            throw new RuntimeException("Part a completar");




//        System.out.println("Seg : " + numSeq + " enviat : " + tempsSegEnviat +
//                " actual : " + tempsActual + " dif : " + difTemps +
//                " finestra congestio " + finestraCongestio);
    }

    private int  random(){
            Random r = new Random();
            int inf = Comms.RTT_INF;
            int sup = Comms.RTT_SUP;
            int res = r.nextInt(sup-inf) + inf;
            return res;
    }
}


public class TSocketRebreCongestioEstilVegas extends TSocket{
    protected CircularQueue cuaRecepcio;

    public TSocketRebreCongestioEstilVegas(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue<>(Comms.MIDA_CUA_RECEPCIO);
    }

    public Object rebre() {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } finally {
            mon.unlock();
        }

    }
```

```
    public void processarMissatge(Segment missatge) {
        mon.lock();
        try{
            throw new RuntimeException("Part a completar");









        } catch (Exception e) { System.out.println(e);
        } finally {
            mon.unlock();
        }
    }
}
```

Connexió-Desconnexió

Tenint en compte el següent diagrama d'estats d'un protocol, completa els mètodes `espera()`, `inicia()` i `processarMissatge(i miss)`. L'exercici es pot interpretar com una simulació de l'establiment de connexió de protocol TCP.

```
/**
                                    +-------------+
                                    | ESTAT_TANC  |-------------
                                    +-------------+             \
                                espera()    |                    \
                                ----------  |                     |   inicia()
                                            V                     | ------------
                                    +-------------+               | envia INICI
                                    | ESTAT_ESP   |               |
                                    +-------------+       +---------------------+
                                           |              | ESTAT_ENV_INICI     |
                                           |              +---------------------+
                                           |                        |
                                reb INICI  |                        | reb INICI_RECON
                            ---------------|                        | ---------------
                            envia INICI_RECON                       |   envia RECON
                                           V                        |
                                +---------------------+             |
                                | ESTAT_REB_INICI     |             |
                                +---------------------+             |
                                           |                        |
                                reb RECON  |                        |
                                -----------|                        |
                                           V                       /
                                    +-------------+               /
                                    | ESTAT_FINAL |<--------------
                                    +-------------+

 */
```

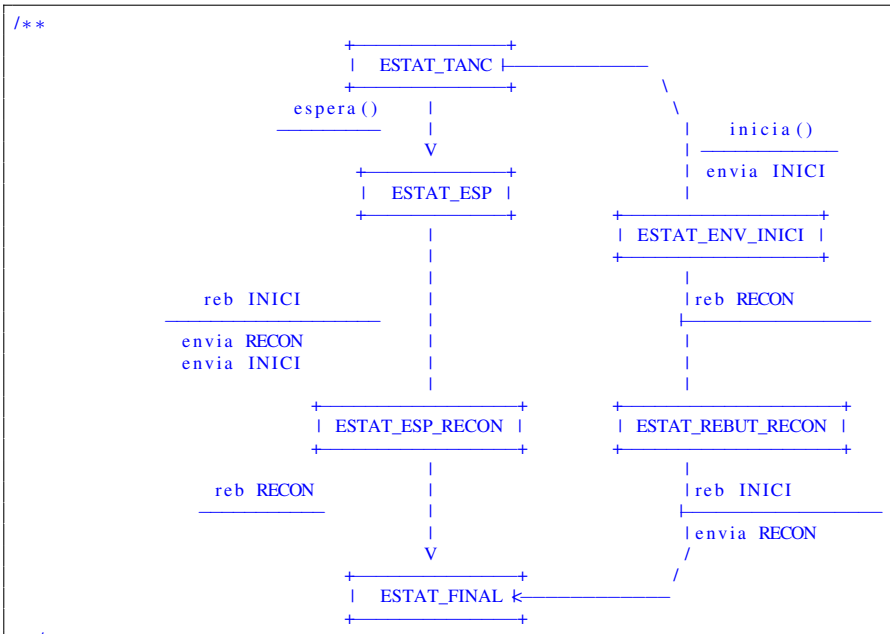
Aspectes a considerar, els mètodes `espera()` i `inicia()` són **bloquejants**, és a dir, **no completen** (retornen) fins que l'estat és `ESTAT_FINAL`.

```java
public abstract class TSocketCanviEstats extends TSocket{

    public TSocketCanviEstats(Xarxa x){
        super(x);
    }

    protected final static int ESTAT_TANC = 0,
                               ESTAT_ESP = 10,
                               ESTAT_ENV_INICI = 20,
                               ESTAT_REB_INICI = 30,
                               ESTAT_FINAL = 40,
                               ESTAT_ESP_RECON = 50,
                               ESTAT_REBUT_RECON = 60;

    public final static int INICI = 100,
                            INICI_RECON = 200,
                            RECON = 300;

    protected int estat;

    public abstract void inicia();
    public abstract void espera();
}
```

```java
public class TSocketCanviEstats1 extends TSocketCanviEstats{

    public TSocketCanviEstats1(Xarxa x)  {
        super(x);
        estat = ESTAT_TANC;
    }

    public void inicia(){
        mon.lock();
        try{
            throw new RuntimeException("Part a completar");















        } catch (Exception ex) { System.out.println(ex); }
        finally{
            mon.unlock();
        }
    }


    public void espera(){
        mon.lock();
        try{
            throw new RuntimeException("Part a completar");











        } finally{
            mon.unlock();
        }
    }

    public void processarMissatge(Object miss){
        mon.lock();
        try {
            switch (estat) {
            case ESTAT_ESP: {
                throw new RuntimeException("Part a completar");






                //break;
            }

            case ESTAT_ENV_INICI: {
                throw new RuntimeException("Part a completar");






                //break;
            }

            case ESTAT_REB_INICI: {
```

```
                throw new RuntimeException("Part a completar");









                //break;
            }
            }
        }  finally {
            mon.unlock();
        }
    }
}
```

Connexió-Desconnexió. Versió 2




```
 */
public class TSocketCanviEstats2 extends TSocketCanviEstats{

    public TSocketCanviEstats2(Xarxa x)  {
        super(x);
        estat = ESTAT_TANC;
    }



    public void inicia(){
        mon.lock();
        try{
            throw new RuntimeException("Part a completar");









        } finally{
            mon.unlock();
        }
    }


    public void espera(){
        mon.lock();
        try{
            throw new RuntimeException("Part a completar");









        } finally{
            mon.unlock();
        }
    }



    public void processarMissatge(Object miss){
        mon.lock();
```

```
        try {
            switch (estat) {
            case ESTAT_ESP: {
                throw new RuntimeException("Part a completar");









                //break;
            }

            case ESTAT_ENV_INICI: {
                throw new RuntimeException("Part a completar");









                //break;
            }

            case ESTAT_REBUT_RECON: {
                throw new RuntimeException("Part a completar");









                //break;
            }

            case ESTAT_ESP_RECON: {
                throw new RuntimeException("Part a completar");









                //break;
            }
            }
        } finally {
            mon.unlock();
        }
    }
}
```

Control Congestió i Flux. Es vol simular el control de flux i congestió d'un protocol de transport. Es considera que en cada segment enviat les dades es limiten a un únic objecte. La quantitat de segments que es poden enviar en un instant determinat, està indicat per la variable `finestraPermesa` de la classe `TSocketEnviarControlFluxCongestio`. Per simplicitat se suposarà que pel control de congestió tenim ajuda de la xarxa sobre la qual s'utilitza el protocol. Aquesta ajuda consisteix a informar als extrems emissors del mínim espai lliure que hi ha en qualsevol de les cues dels nodes en tots els possibles camins fins als corresponents extrems receptors. Se suposa que si la `finestraPermesa` en l'emissor és menor que aquest mínim no hi haurà pèrdues. La manera que té la xarxa d'informar als extrems emissors és enviant un `Segment` de tipus `CONG`, que conté el valor mínim de l'espai lliure en les cues en el camp `finestra`.

El control de flux es fa com al TCP, on el receptor informa a l'emissor de l'espai lliure que hi ha en la seva cua de recepció.

- Per simplicitat **totes** les variables enteres poden ser incrementades sense límit.
- L'atribut `numeroSequencia` de la classe `Segment`, indica el número de seqüència si el segment és de dades, i el número del segment que reconeix si és un segment d'ACK.
- La cua circular en la classe `TSocketRebreControlFluxCongestio` guarda les dades rebudes. Els mètodes de que disposa són els habituals.
- El mètode `enviar(Object c)` és bloquejant si enviar un enter (segment) suposa superar la `finestraPermesa`.
- El mètode `int rebre()` és bloquejant si la cua circular és buida.

```
public class TSocketEnviarControlFluxCongestio extends TSocket{

    protected int ultimSegmentEnviat, ultimSegmentReconegut, finestraRecepcio,
                  finestraCongestio, finestraPermesa;


    public TSocketEnviarControlFluxCongestio(Xarxa x) {
        super(x);
        finestraRecepcio = Comms.MIDA_CUA_RECEPCIO;
        finestraPermesa = 1;
        new Thread(new SimulacioCongestioXarxa()).start();

    }

    public void enviar(Object c) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");




        } finally {
            mon.unlock();
        }
    }

    @Override
    public void processarMissatge(Object segment) {
        mon.lock();
            if(((Segment)segment).getTipus()==Comms.ACK){
                    throw new RuntimeException("Part a completar");



            }else if(((Segment)segment).getTipus()==Comms.CONG){
                   throw new RuntimeException("Part a completar");


            }
            actualitzarFinestraPermesa();
            appCV.signalAll();
        mon.unlock();
    }
```

```
    private void actualitzarFinestraPermesa(){
                throw new RuntimeException("Part a completar");



    }

    class SimulacioCongestioXarxa implements Runnable {

        public void run() {
            while (true) {
                try {
                    Thread.sleep((long)(Math.random()*5000));
                } catch (InterruptedException ex) {
                    Logger.getLogger(TSocketEnviarControlFluxCongestio.class.getName()).log(Level.SEVERE, null, ex);
                }
                int finesCong = (int)(Math.random()*50);
                Segment segment = new Segment(Comms.CONG,0,finesCong,0);

                processarMissatge(segment);
            }
        }
    }
}
```

```
public class TSocketRebreControlFluxCongestio extends TSocket{

    protected int numSeqEsperat;
    protected CircularQueue cuaRecepcio;

    public TSocketRebreControlFluxCongestio(Xarxa x) {
        super(x);
        cuaRecepcio = new CircularQueue<>(Comms.MIDA_CUA_RECEPCIO);
    }

    public Object rebre() {
        mon.lock();
        while (cuaRecepcio.empty()) {
            try {
                appCV.await();
            } catch (InterruptedException ex) {}
        }

        Object res = cuaRecepcio.get();
        mon.unlock();
        return res;
    }


    public void processarMissatge(Object missatge) {
        mon.lock();
        try {
            throw new RuntimeException("Part a completar");









        } finally {
            mon.unlock();
        }
    }
}
```