**REPASO PARCIAL 2**

8. Partimos de la siguiente especificación formal de un recurso compartido:

**OPERACIONES**

**ACCIÓN A:**
**ACCIÓN B:**
**ACCIÓN C:**

**SEMÁNTICA**
**DOMINIO:**
**TIPO:** $Binario = (p : \mathbb{B} \times q : \mathbb{B})$

**INICIAL:** $self.p \wedge self.q$

**CPRE:** Cierto
**A()**
**POST:** $self.p \wedge (self.q = self^{pre}.q)$

**CPRE:** Cierto
**B()**
**POST:** $\neg self.p \wedge self.q$

**CPRE:** $self.p \wedge self.q$
**C()**
**POST:** $\neg self.p \wedge \neg self.q$

Alguien ha decidido implementarlo mediante monitores de la siguiente manera:

```
class Binario {
    private boolean p = true;
    private boolean q = true;
    private Monitor mutex = new Monitor();
    private Monitor.Cond cond_p = mutex.newCond();
    private Monitor.Cond cond_q = mutex.newCond();

    void a() {
        mutex.enter();
        p = true;
        cond_p.signal();
        mutex.leave();
    }
    void b() {
        mutex.enter();
        p = false;
        q = true;
        cond_q.signal();
        mutex.leave();
    }
    void c() {
        mutex.enter();
        if (!p) { cond_p.await(); }
        if (!q) { cond_q.await(); }
        p = false;
        q = false;
        mutex.leave();
    }
}
```

Se pide señalar la respuesta correcta:
(a) Es una implementación correcta del recurso compartido.
(b) El código puede provocar que se ejecute c() en un estado en el que no se cumple su CPRE
(c) El código de c está incompleto: falta el código de desbloqueos.

9. ¿Debería permitirse a un *thread* invocar una operación de `await` sobre un objeto de clase `Monitor.Cond` generado a partir de un objeto de la clase `Monitor` sin previamente haber invocado el método `enter`?
   (a) Sí
   (b) No

10. **Se define una clase *servidor* `P` y dos clases *cliente* `P1` y `P2` que se comunican a través de los canales de comunicación `petA` y `petB` (se muestran las partes relevantes del código para este problema).**

```
class P implements CSProcess {
  public void run() {
    // Estado del recurso
    boolean q = true;
    boolean r = true;

    final int A = 0;
    final int B = 1;

    final Guard[] entradas =
      {petA.in(), petB.in()};

    final Alternative servicios =
      new Alternative (entradas);

    final boolean[] sincCond =
      new boolean[2];

    while (true) {
      sincCond[A] = q || r;
      sincCond[B] = q && r;

      switch (servicios.fairSelect(sincCond)) {
        case A:
          petA.in().read();
          q = !q;
          break;
        case B:
          petB.in().read();
          r = !r;
          break;
      }
    }
  }
}
```

```
class P1 implements CSProcess {
  public void run() {
    while (true) {
      petA.out().write(null);
    }
  }
}

class P2 implements CSProcess {
  public void run() {
    while (true) {
      petB.out().write(null);
    }
  }
}
```

Dado un programa concurrente con tres procesos p, p1 y p2 de las clases P, P1 y P2.
Se pide decir cuál de las siguientes afirmaciones es la correcta:
(a) Es seguro que los tres procesos se van a bloquear.
(b) Es seguro que p2 acabará bloqueándose, pero p y p1 podrían seguir ejecutando indefinidamente.
(c) Es posible que p1 se bloquee pero en ese caso p, y p2 podrían seguir ejecutando indefinidamente.
(d) Ninguna de las respuestas anteriores.

11.

Está a punto de inaugurarse el primer centro comercial en el que la compra la realizan robots: e-Qea. Los compradores envían sus listas de la compra y los robots inician el recorrido. El centro comercial lo componen $N$ (un número fijo) secciones adyacentes. Los robots inician la compra entrando en la sección 0, avanzan a la sección adyacente (0, 1, ...) cuando han acumulado los productos requeridos de esa sección y terminan saliendo de la sección $N-1$.

El centro e-Qea tiene un problema estructural (literalmente): cada sección soporta un peso máximo ($P$). Esto significa que cuando un robot con un determinado peso ($p$) quiere avanzar, debe esperar hasta que el incremento de peso que provoca no ponga en peligro la estructura (si el peso actual de la sección adyacente es *peso* entonces un robot con peso $p$ no puede avanzar si $peso + p > P$).

El siguiente recurso compartido gestiona el movimiento de robots por e-Qea:

**C-TAD** ControleQea

**OPERACIONES**

**ACCIÓN** avanzar: $TipoSección[e] \times \mathbb{N}[e]$

**SEMÁNTICA**
**DOMINIO:**
**TIPO:** $ControleQea = TipoSección \nrightarrow \mathbb{N}$
**TIPO:** $TipoSección = \{0, 1, \ldots, N\}$
**INVARIANTE:** $\forall s \in TipoSección \bullet self(i) \leq P$

**INICIAL:** $\forall s \in TipoSección \bullet self(i) = 0$

**PRE:** Cierto
**CPRE:** $s < N \Rightarrow \text{self}(s) + p \leq P$
**avanzar(s,p)**
**POST:** $s = 0 \Rightarrow \text{self} = \text{self}^{pre} \oplus \{s \mapsto \text{self}^{pre}(s) + p\}$
$s > 0 \Rightarrow \text{self} = \text{self}^{pre} \oplus \{s-1 \mapsto \text{self}^{pre}(s-1) - p, s \mapsto \text{self}^{pre}(s) + p\}$

Los *threads* que controlan los robots ejecutan la operación *avanzar*$(s, p)$ cuando el robot quiere avanzar de la sección $s-1$ a la sección $s$ portando un peso $p$ (con $s = 0$ se indica la entrada al centro y con $s = N$ la salida del mismo).

**Se pide** implementar el recurso compartido en Java usando la librería JCSP.

MONITORES:

```
public class ControleQea {

  private Monitor mutex;
  private List<PetAvanzar> listaPet;
  final static int P; // peso máximo
  final static int N; // tamaño máximo
  private int[] tipoSeccion;

  public ControleQea(int peso, int tam) {
    P = peso;
    N = tam;
    tipoSeccion = new int[tam];
    mutex = new Monitor();
    listaPet = new ArrayList<>();
  }

  private class PetAvanzar {
    private int s, p;
    private Monitor.Cond cond;
    public PetAvanzar(int s, int p) {
      this.s = s;
      this.p = p;
      cond = mutex.newCond();
    }
  }

  public void avanzar(int s, int p) {
    mutex.enter();
    // IF(!PRE) -> Excepcion → NO EXISTE en este enunciado
    // IF(!CPRE) -> Añado a la lista
    if(s >= N || tipoSeccion[s] + p > P) {
      PetAvanzar pet = new PetAvanzar(s,p);
      listaPet.add(pet);
      pet.cond.await();
    }
    // POST
    if(s == 0) {
      tipoSeccion[s] += p;
    } else {
      tipoSeccion[s-1] -= p;
      tipoSeccion[s] += p;
    }
```

```
    // Desbloqueo
    int pos = 0; boolean desbloqueo = false;
    while(pos < listaPet.size() && !desbloqueo) {
      if(listaPet.get(pos).s < N &&
         tipoSeccion[listaPet.get(pos).s] + listaPet.get(pos).p <= P) {

         listaPet.get(i).cond.signal;
         lista.remove(i);
         desbloqueo = true;
      } else
        pos++;
    }
    mutex.leave();
  }
}
```

JCSP:

```
public class ControleQeaCSP extends CSProcess {

  Any2OneChannel chAvanzar;
  private List<PetAvanzar> listaPet;
  final static int P; // peso máximo
  final static int N; // tamaño máximo
  private int[] tipoSeccion;

  public ControleQea(int peso, int tam) {
    P = peso;
    N = tam;
    tipoSeccion = new int[tam];
    listaPet = new ArrayList<>();
    chAvanzar = Channel.any2one();
  }

  private class PetAvanzar {
    private int s, p;
    private One2OneChannel cond;
    public PetAvanzar(int s, int p) {
      this.s = s;
      this.p = p;
      cond = Channel.one2one();
    }
  }

  public void avanzar(int s, int p) {
    PetAvanzar pet = new PetAvanzar(s,p);
    chAvanzar.out().write(pet);
    pet.cond.in().read();
  }
```

```
  public void run() {
    List<PetAvanzar> lista = new ArrayList<>();
    Guard entrada = chAvanzar.in();
    Alternative servicios = new Alternative(entrada);
   while(true) {
    PetAvanzar pet = (PetAvanzar) chAvanzar.in().read();
    if(s >= N || tipoSeccion[s] + p > P) {
      lista.add(pet);
    } else if(pet.s == 0 && tipoSeccion[pet.s] + pet.p <= P) {
      tipoSeccion[s] += p;
      pet.cond.out().write(null);
    } else if(pet.s > 0 && tipoSeccion[pet.s] + pet.p <= P) {
      tipoSeccion[pet.s-1] -= pet.p;
      tipoSeccion[pet.s] += pet.p;
      pet.cond.out().write(null);
    }

   // Desbloqueo peticiones avanzadas
    int pos = 0;
    while(pos < listaPet.size()) {
      if(listaPet.get(pos).s < N &&
         tipoSeccion[listaPet.get(pos).s] + listaPet.get(pos).p <= P) {
         listaPet.get(i).cond.write(null);
         lista.remove(i);
       } else
         pos++;
     } // while
   } // while(true)
  } // run
} // class
```

12. Se define una clase servidor P y tres clases cliente P1, P2 y P3 que se comunican a través de los canales de comunicación petA, petB y petC (se muestran las partes relevantes del código para este problema).

```
class P implements CSProcess {
  public void run() {
    boolean q = false;
    boolean r = false;
    boolean s = false;
    final int A = 0;
    final int B = 1;
    final int C = 2;
    final Guard[] entradas =
        {petA.in(), petB.in(),petC.in()};
    final Alternative servicios =
        new Alternative (entradas);
    final boolean[] sincCond =
        new boolean[3];

    while (true) {
      sincCond[A] = !(q || r);
      sincCond[B] = !(q && r);
      sincCond[C] = !s;
      int sel = servicios.fairSelect(sincCond);
      switch (sel) {
      case A:
        petA.in().read();
        q = !q;
        break;
      case B:
        petB.in().read();
        r = !r;
        break;
      case C:
        petC.in().read();
        break;
      }
    }
  }
}
```

```
class P1 implements CSProcess {
  public void run() {
    while (true) {
      petA.out().write(null);
    }
  }
}

class P2 implements CSProcess {
  public void run() {
    while (true) {
      petB.out().write(null);
    }
  }
}

class P3 implements CSProcess {
  public void run() {
    while (true) {
      petC.out().write(null);
    }
  }
}
```

Dado un programa concurrente con cuatro procesos p, p1, p2 y p3 de las clases P, P1, P2 y P3.
Se pide marcar cuál de las siguientes afirmaciones es la correcta:
(a) Es seguro que los cuatro procesos se van a bloquear.
(b) Es seguro que p1 y p2 acabarán bloqueándose, pero p y p3 podrían seguir ejecutando indefinidamente.
(c) Es posible que p3 se bloquee pero en ese caso p, p1 y p2 podrían seguir ejecutando indefinidamente.
(d) Ninguna de las otras respuestas.

13. Dado el sistema de la pregunta 5 implementado con JCSP. Se pide decir cuál de las siguientes afirmaciones es la correcta:
(a) El array sincCond contiene la evaluación de las CPREs de las operaciones A, B y C.
(b) Cuando fairSelect devuelve un valor sel se cumple que sincCond[sel] == true.
(c) Ambas son correctas

4. Una pequeña variación del típico problema del productor-buffer-consumidos es el llamado “buffer de pares e impares”. La idea es que en la operación de extracción se permite decir si queremos retirar un número par o un número impar. Se pide implementar el recurso compartido usando paso de mensajes síncrono, mediante la librería JCSP, a partir de la siguiente especificación:

**C-TAD BufferPI**

**OPERACIONES**
**ACCIÓN** Poner: *Tipo_Dato[e]*
**ACCIÓN** Tomar: *Tipo_Paridad[e]* × *Tipo_Dato[s]*

**SEMÁNTICA**
**DOMINIO:**
**TIPO:** *Tipo_Buffer_PI* = Secuencia(*Tipo_Dato*)
*Tipo_Paridad* = *par*|*impar*
*Tipo_Dato* = $\mathbb{N}$
**INVARIANTE:** $\forall b \in Tipo\_Buffer\_PI \bullet \text{Longitud}(b) \leq MAX$

**DONDE:** $MAX = ...$
**INICIAL:** $\text{Longitud}(self) = 0$

***CPRE*:** *El buffer no está lleno*
**CPRE:** $\text{Longitud}(self) < MAX$
**Poner(d)**
***POST*:** *Añadimos un elemento al buffer*
**POST:** $l = \text{Longitud}(self^{pre}) \wedge \text{Longitud}(self) = l+1 \wedge self(l+1) = d^{pre} \wedge self(1..l) = self^{pre}$

***CPRE*:** *El buffer no está vacío y el primer dato preparado para salir es del tipo que requerimos*
**CPRE:** $\text{Longitud}(self) > 0 \wedge Concuerda(self(1), t)$
**DONDE:** $Concuerda(d, t) \equiv (d \bmod 2 = 0 \leftrightarrow t = par)$
**Tomar(t, d)**
***POST*:** *Retiramos el primer elemento del buffer*
**POST:** $l = \text{Longitud}(self^{pre}) \wedge self^{pre}(1) = d \wedge self = self^{pre}(2..l)$

La solución consistirá en una clase Gestor_PI con la siguiente estructura:

```
static class Gestor_PI implements CSProcess {
  private static final int PAR = 0;
  private static final int IMPAR = 1;
  // A: declaraciones
  public Gestor_PI() {
    // B: inicializacion
  }
  public void poner(int d) {
    // C: op. poner
  }
  public int tomar(int t) {
    // D: op. tomar
  }
    public void run() {
    // E: cod. servidor
  }
}
```

```
static class Gestor_PI implements CSProcess {
  private static final int PAR = 0;
  private static final int IMPAR = 1;
  // A: declaraciones
  private enum TipoParidad = {PAR, IMPAR};
  private List<Integer> bufferPI;
  private final static int MAX = 1000;

  Any2OneChannel chPoner;
  Any2OneChannel chTomar;

  public Gestor_PI() {
    // B: inicializacion
    bufferPI = new ArrayList<>();
    chPoner = Channel.any2one();
    chTomar = Channel.any2one();
  }

  private class PetTomar {
    private int t;
    private One2OneChannel cond;

    public PetTomar(int t) {
      this.t = t;
      cond = Channel.one2one();
    }
  }

  public void poner(int d) {
    // C: op. poner
    chPoner.out().write(d);
  }
  public int tomar(int t) {
    // D: op. tomar
    PetTomar pet = new PetTomar(t);
    chTomar.out().write(pet);
    return (int)pet.cond.in().read();
  }
```

```
  public void run() {
    // E: cod. servidor
    List<PetTomar> peticiones = new ArrayList<>();
    Guard[] entradas = {chPoner.in(), chTomar.in()};
    Alternative servicios = new Alternative(entradas);

    final int PONER = 0;
    final int TOMAR = 1;

    boolean[] sincCond = new boolean[2];

    while(true) {
      sincCond[PONER] = bufferPI.size() < MAX;
      sincCond[TOMAR] = bufferPI.size() > 0;

      switch(servicios.fairSelect(sincCond)) {
        case PONER:
          int d = (int) chPoner.in().read();
          bufferPI.add(d);
          break;

        case TOMAR:
          PetTomar pet = (PetTomar) chTomar.in().read();
          if(pet.t == PAR && bufferPI.get(0)%2 == 0 ||
             pet.t == IMPAR && bufferPI.get(0)%2 != 0) {
             pet.cond.out().write(bufferPI.get(0));
             bufferPI.remove(0);
          } else {
            peticiones.add(pet);
          }
          break;
      } //switch

      // Desbloqueo de las peticiones aplazadas
      int pos = 0;
      while(pos < peticiones.size()) {
        PetTomar pet = peticiones.get(pos);
        if((pet.t == PAR && bufferPI.get(0)%2 == 0) ||
              (pet.t == IMPAR && bufferPI.get(0)%2 != 0)) {
          pet.cond.out().write(bufferPI.get(0));
          bufferPI.remove(0);
          peticiones.remove(pos);
        } else
          pos++;
      }
    } //while(true)

  } //run
}
```