1. Usa ProcessBuilder para lanzar un hipotético proceso localizado en la ruta C:\Windows\System32\Mspaint.exe (1pto)

```
import java.io.IOException;
public class Principal {
  public static void main(String[] args) {
    ProcessBuilder proceso;
    proceso = new ProcessBuilder("C:\Windows\System32\Mspaint.exe");
    try {
      proceso.start();
      System.out.println("Proceso lanzado con éxito");
    } catch (IOException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

1.1 Supon que este proceso permite la posibilidad de guardar el dibujo de antemano pasando un parámetro a processbuilder, añade la posibilidad de guardar el dibujo en el fichero DIBUJO.JPG) (1 pto)

```
import java.io.IOException;
public class Principal3 {
  public static void main(String[] args) {
    ProcessBuilder proceso;
    proceso = new ProcessBuilder("C:/Windows/System32/notepad.exe", "agenda.txt");
    try {
      proceso.start().waitFor();
      System.out.println("Proceso lanzado con éxito");
    } catch (IOException | InterruptedException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

2. TRABAJO COOPERATIVO (ALQUILER DE COCHES)

2.1 una compañía de alquiler de coches tiene coches disponibles para reserva, y esos coches se usan conforme se devuelven.

Los coches están categorizados por tipo, coche tipo 1, coche tipo2, coche tipo3, coche tipo 4, coche tipo5.

El sistema registra la entrada del tipo del coche. Vamos a hacer un generador aleatorio de entrada de coches de tipo entero.

El sistema es FIFO, de forma que el primer tipo de coche que entra es el primer tipo de coche que se sirve para alquiler, no existe ningún tipo de demanda por parte del usuario, simplemente coge el primero que hay en la cola

a) Desarrolla la clase BufferCoche basada en la clase LinkedList y con dos métodos (poner y sacar) que controlen la sincronización y el bloqueo en caso de no haber existencias. El método poner se encarga de añadir un nuevo tipo de coche a la lista y el método sacar que devuelva el primer valor de la lista y elimine el tipo de coche de la misma (1 pto)

BUFFER DE COCHE

```
import java.util.LinkedList;
import java.util.Queue;
public class BufferCoche
{
  private Queue<Integer> cola;
  //private LinkedList<integer> cola;

  public BufferCoche()
  {
    cola = new LinkedList<Integer>();
  }

  public synchronized void poner(Integer c)
  {
    cola.add(c);
    notify();
  }
  public synchronized Integer sacar()
  {
    while (cola.isEmpty())
    {
      try
      {
        wait();
      }
      catch (InterruptedException e)
      {
        System.out.println(e.getMessage());
      }
    }
    if (!cola.isEmpty())
    {
      Integer tipo = cola.remove();
      return tipo;
    }
```

```
      else
      {
        return null;
      }
    }
}
```

b) Desarrolla una clase hilo llamado Devolucion que pueda identificar de forma univoca cada uno de los hilos lanzados, que de forma aleatoria genere un tipo de coche (del 1 al 5) e introduzca ese nuevo tipo de coche en el BufferCoche. (1pto)

DEVOLUCION COCHE

```
public class Devolucion implements Runnable
{
  private BufferCoche buffer;
  private static int contador = 0;
  private Thread hilo;

  public Devolucion(BufferCoche buffer)
  {
    contador ++;
    this.buffer = buffer;
    hilo = new Thread(this, "Devolucion n"+contador);
    hilo.start();
  }

  public void run()
  {
    int TipoCoche;
    TipoCoche = (int) (Math.random()*5+1);
    System.out.println(hilo.getName() + " acaba de devolver un coche del tipo " +
TipoCoche);
    buffer.poner(TipoCoche);
  }
}
```

c) Desarrolla una clase hilo llamada Reserva que identifique de forma univoca cada uno de los hilos lanzados, que cuando sea lanzado reserve el primer coche que se encuentre en la lista.

RESERVA COCHE (1pto)

```
public class Reserva implements Runnable
{
  private Thread hilo;
  private BufferCoche buffer;
  private static int contador = 0;
  public Reserva(BufferCoche buffer)
```

```
  {
    contador++;
    hilo = new Thread(this, "ReservaN" + contador);
    this.buffer = buffer;
    hilo.start();
  }

  public void run()
  {
    ReservarUnCoche();
  }
  public void ReservarUnCoche()
  {
    Integer TipoCoche = buffer.sacar();
    if (TipoCoche != null)
    {

      System.out.println(hilo.getName() + " acaba de reservar un coche del tipo N " +
TipoCoche);
    }
    else
    {
      System.out.println("Agotado tiempo de espera y no hay coches para reservar");
    }
  }
}
```

LANZADOR Crea una clase que lance alternativamente lance 3 veces el proceso Devolucion y el proceso Reserva. Acuérdate de instanciar la cola común de elementos (buffer) (0,5ptos)

```
public class Lanzador
{
  public static void main(String[] args)
  {
    BufferCoche buffer = new BufferCoche();
    new Devolucion(buffer);
    new Reserva(buffer);
    new Devolucion (buffer);
    new Reserva (buffer);
    new Devolucion (buffer);
    new Reserva (buffer);
  }
}
```

2.2 MODIFICA EL EJERCICIO ANTERIOR PARA CONSIDERAR MARCAS DE COCHE EN VEZ DE TIPOS DE COCHES, MANTENIENDO LA MISMA ESTRUCTURA TIPO. (1,5 ptos)

BUFFER

```
import java.util.LinkedList;
import java.util.Queue;
public class BufferCoche2 {
  private Queue<String> cola;
  public BufferCoche2() {
    cola = new LinkedList<String>();
  }
  public synchronized void poner(String marca) {
    cola.add(marca);
    notify();
  }
  public synchronized String sacar() {
    while (cola.isEmpty()) {
      try {
        wait();
      } catch (InterruptedException e) {
        System.out.println(e.getMessage());
      }
    }
    if (!cola.isEmpty()) {
      String marca = cola.remove();
      return marca;
    } else {
      return null;
    }
  }
}
```

DEVOLUCION

```
public class Devolucion implements Runnable {
   private BufferCoche buffer;
   private static int contador = 0;
   private Thread hilo;
   // Lista de marcas de coches
   private static final String[] marcas = {"Ford", "BMW", "Audi", "Toyota", "Mercedes"};
   public Devolucion(BufferCoche buffer) {
      contador++;
      this.buffer = buffer;
      hilo = new Thread(this, "Devolucion n" + contador);
      hilo.start();
   }
   public void run() {
      // Generar una marca aleatoria de coche
      String marcaCoche = marcas[(int) (Math.random() * marcas.length)];
      System.out.println(hilo.getName() + " acaba de devolver un coche de marca " +
marcaCoche);
      buffer.poner(marcaCoche);
   }
}
```

RESERVA

```
public class Reserva implements Runnable {
   private Thread hilo;
   private BufferCoche buffer;
   private static int contador = 0;
   public Reserva(BufferCoche buffer) {
      contador++;
      hilo = new Thread(this, "ReservaN" + contador);
      this.buffer = buffer;
```

```
        hilo.start();
    }
    public void run() {
        ReservarUnCoche();
    }
    public void ReservarUnCoche() {
        String marcaCoche = buffer.sacar();
        if (marcaCoche != null) {
            System.out.println(hilo.getName() + " acaba de reservar un coche de marca " +
marcaCoche);
        } else {
            System.out.println("Agotado tiempo de espera y no hay coches para reservar");
        }
    }
}
```

LANZADOR

Sin cambios

3. HAZ UN SISTEMA CLIENTE/SERVIDOR DE UNA UNICA CONSULTA CON UN UNICO HILO SIMULTANEO PARA CONSULTAR EN UN SERVIDOR 2 POSIBLES DATOS: 1 NUMERO DE USUARIOS DEL SISTEMA O 2 NUMERO DE CURSOS DISPONIBLES.

    a) Crea una clase Servidor que escuche en el puerto 12233 con la intención de establecer un Socket con un cliente. El servidor debe tener dos parámetros: parámetro uno, NumeroUsuarios con valor = 15 y NumeroCursos = 25. Una vez aceptado se queda esperando un mensaje del cliente y posteriormente envia al cliente una información:

    Si recibe un 1 envia el contenido de la variable NumeroUsuarios y si recibe un 2 envia el contenido de la variable NumeroCursos. Cualquier otra información, manda un mensaje diciendo, no se encuentra disponible esa opción en el menú. Posteriormente cierra la conexión. (1,5 ptos)

SERVIDOR

```
import java.net.ServerSocket;
import java.net.Socket;
import java.io.InputStream;
```

```
import java.io.OutputStream;
public class Servidor {
   public static void main(String[] args) {
      // Definir las variables de los parámetros
      final int NumeroUsuarios = 15;
      final int NumeroCursos = 25;
      try {
         // Crear un ServerSocket que escuche en el puerto 12233
         ServerSocket serverSocket = new ServerSocket(12233);
         System.out.println("Servidor escuchando en el puerto 12233...");
         // Esperar a que un cliente se conecte
         Socket clientSocket = serverSocket.accept();
         System.out.println("Cliente conectado.");
         // Recibir el mensaje del cliente
         InputStream inputStream = clientSocket.getInputStream();
         byte[] buffer = new byte[1024];
         int bytesRead = inputStream.read(buffer);
         String mensajeCliente = new String(buffer, 0, bytesRead).trim();
         System.out.println("Mensaje recibido del cliente: " + mensajeCliente);
         // Preparar el mensaje a enviar al cliente
         String mensajeServidor = "";
         if ("1".equals(mensajeCliente)) {
            mensajeServidor = "Número de usuarios: " + NumeroUsuarios;
         } else if ("2".equals(mensajeCliente)) {
            mensajeServidor = "Número de cursos: " + NumeroCursos;
         } else {
            mensajeServidor = "No se encuentra disponible esa opción en el menú.";
         }
         // Enviar el mensaje de respuesta al cliente
         OutputStream outputStream = clientSocket.getOutputStream();
         outputStream.write(mensajeServidor.getBytes());
```

```
            outputStream.flush(); // Asegurarse de que los datos se envíen
        inmediatamente
            System.out.println("Mensaje enviado al cliente: " + mensajeServidor);
            // Cerrar la conexión con el cliente y el servidor
            clientSocket.close();
            System.out.println("Conexión con el cliente cerrada.");
            serverSocket.close();
            System.out.println("Servidor cerrado.");
        } catch (Exception e) {
            System.out.println("Error en el servidor: " + e.getMessage());
        }
    }
}
```

b) Desarrolla una clase Cliente que realice una petición de establecimiento de Socket sobre la ip 127.0.0.1 y al puerto del servidor. Muestre al usuario 2 opciones:

1 NUMERO DE USUARIOS DEL SISTEMA

2 NUMERO DE CURSOS DISPONIBLES

Que el usuario por teclado escriba la opción y esta se envie al servidor. El cliente se queda esperando la respuesta del servidor y luego cierra la conexión.

CLIENTE

```
import java.net.Socket;
import java.io.InputStream;
import java.io.OutputStream;
import java.util.Scanner;
public class Cliente {
  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    try {
      // Conexión al servidor
```

```
            Socket socket = new Socket("127.0.0.1", 12233);
            System.out.println("Conectado al servidor en 127.0.0.1:12233");
            // Mostrar el menú de opciones
            System.out.println("Seleccione una opción:");
            System.out.println("1. Número de usuarios del sistema");
            System.out.println("2. Número de cursos disponibles");
            System.out.print("Ingrese su opción: ");
            String opcion = scanner.nextLine(); // Leer la opción del usuario
            // Enviar la opción al servidor
            OutputStream outputStream = socket.getOutputStream();
            outputStream.write(opcion.getBytes());
            outputStream.flush();
            System.out.println("Opción enviada al servidor.");
            // Recibir la respuesta del servidor
            InputStream inputStream = socket.getInputStream();
            byte[] buffer = new byte[1024];
            int bytesRead = inputStream.read(buffer);
            String respuestaServidor = new String(buffer, 0, bytesRead);
            System.out.println("Respuesta del servidor: " + respuestaServidor);
            // Cerrar la conexión
            socket.close();
            System.out.println("Conexión cerrada.");
        } catch (Exception e) {
            System.out.println("No se pudo conectar al servidor: " + e.getMessage());
        }
    }
}
```