EJEMPLO DE EXAMEN.

1) **(1.5 punto)**

Explica la diferencia fundamental entre un socket datagram (UDP) y un socket stream (TCP) en términos de orientación a conexión. ¿Por qué se dice que UDP es "sin conexión" y TCP es "orientado a conexión"?

UDP es "sin conexión" porque no requiere establecer una conexión antes de enviar datos. Simplemente envía paquetes (datagramas) al destino sin verificar si el receptor está listo o disponible.

TCP es "orientado a conexión" porque establece una conexión entre el emisor y el receptor antes de enviar datos. Esto implica un intercambio inicial y garantiza que los datos se entreguen en orden y sin errores.

2) **(1.5 punto)**

Imagina que estás diseñando una aplicación de chat. ¿Elegirías utilizar UDP o TCP? Justifica tu decisión basándote en las características de ambos protocolos.

Se elegiría TCP para una aplicación de chat porque la confiabilidad y la entrega ordenada de los mensajes son críticas. En un chat, es importante que los mensajes lleguen completos y en el orden correcto, lo cual TCP garantiza.

UDP no sería adecuado porque, aunque es más rápido, no garantiza la entrega de los mensajes ni su orden, lo que podría resultar en mensajes perdidos o desordenados, afectando la experiencia del usuario.

3) **(2 PUNTOS)**

¿Cómo se gestiona la concurrencia en un servidor cuando múltiples clientes se conectan al mismo tiempo? Explica cómo se utilizan los conceptos de synchronized, wait, notify y notifyAll

En un servidor concurrente, cada cliente es atendido por un **hilo separado**, lo que permite manejar múltiples conexiones simultáneamente sin que un cliente bloquee a los demás. Para gestionar el acceso a recursos compartidos (como variables o estructuras de datos), se utilizan los siguientes mecanismos de sincronización en Java:

1. **synchronized**:
    - Bloquea un método o bloque de código para que solo un hilo pueda ejecutarlo a la vez.
    - Ejemplo: Si varios hilos intentan modificar una lista de clientes conectados, synchronized asegura que solo uno lo haga en cada momento.
2. **wait**:

- Hace que un hilo espere hasta que otro hilo lo notifique.
- Útil para pausar un hilo hasta que se cumpla una condición (por ejemplo, esperar a que un recurso esté disponible).

3. **Notify y notifyAll**:
   - notify: Despierta a un solo hilo que esté en espera.
   - notifyAll: Despierta a todos los hilos en espera.
   - Se usan para avisar a los hilos que un recurso está disponible o que una condición ha cambiado.

, y cómo el servidor emplea hilos separados para evitar que un cliente bloquee la comunicación con otros? (1 punto)

Funcionamiento del servidor:

- Cuando un cliente se conecta, el servidor crea un **nuevo hilo** para atenderlo (HiloEscuchador), permitiendo que otros clientes se conecten sin ser bloqueados.
- Si varios hilos necesitan acceder a un recurso compartido (como una base de datos o una lista de conexiones), se usa synchronized para evitar condiciones de carrera.
- Si un hilo necesita esperar (por ejemplo, a que lleguen datos), usa wait. Cuando el recurso está listo, otro hilo lo notifica con notify o notifyAll.

4) (2 puntos)

¿Qué pasos son necesarios para implementar un servicio RMI en Java?

Para implementar un servicio RMI en Java, se deben seguir los siguientes pasos:

1. Crear una interfaz remota que extienda java.rmi.Remote y defina los métodos que se invocarán remotamente.
2. Implementar la interfaz remota en una clase que extienda UnicastRemoteObject.
3. Crear un registro de objetos remotos utilizando LocateRegistry.createRegistry().
4. Registrar el objeto remoto en el registro utilizando rebind().
5. En el cliente, obtener el stub del objeto remoto utilizando lookup() y luego invocar los métodos remotos.

5) (1 punto)

¿Qué ventajas tiene JSON sobre XML en la transferencia de datos en aplicaciones distribuidas?

JSON es más ligero, más fácil de leer y escribir, y tiene una sintaxis más simple que XML. Además, JSON es nativamente compatible con JavaScript, lo que lo hace ideal para aplicaciones web modernas.

6) (2 puntos)

Define un fichero JSON que represente una agenda de teléfonos. Explica los campos que incluirías y su propósito.

```
{
 "agenda": [
  {
   "nombre": "Juan Pérez",
   "telefono": "600123456",
   "email": "juan.perez@example.com",
   "direccion": "Calle Falsa 123, Madrid",
   "notas": "Amigo de la universidad"
  },
  {
   "nombre": "María Gómez",
   "telefono": "655987654",
   "email": "maria.gomez@example.com",
   "direccion": "Avenida Real 45, Barcelona",
   "notas": "Contacto de trabajo"
  },
  {
   "nombre": "Carlos Ruiz",
   "telefono": "699555444",
   "email": "carlos.ruiz@example.com",
   "direccion": "Plaza Mayor 7, Valencia",
```

```
    "notas": "Vecino"
  }
 ]
}
```

Explicación de los campos:

1. **agenda**:
    - Es el contenedor principal que almacena una lista de contactos.
    - Tipo: Array de objetos.
2. **nombre**:
    - Almacena el nombre completo del contacto.
    - Tipo: String.
    - Ejemplo: "Juan Pérez".
3. **telefono**:
    - Contiene el número de teléfono del contacto.
    - Tipo: String (se usa String para evitar problemas con formatos internacionales o caracteres especiales como "+").
    - Ejemplo: "600123456".
4. **email**:
    - Almacena la dirección de correo electrónico del contacto.
    - Tipo: String.
    - Ejemplo: "juan.perez@example.com".
5. **direccion**:
    - Contiene la dirección física del contacto.
    - Tipo: String.
    - Ejemplo: "Calle Falsa 123, Madrid".
6. **notas**:
    - Permite añadir información adicional sobre el contacto, como su relación o detalles relevantes.
    - Tipo: String.

      Ejemplo: "Amigo de la