Leé con cuidado el enunciado y por lo menos dos veces para resolver lo pedido. Pensá bien la estrategia de resolución antes de comenzar el desarrollo de lo que te solicitan. El objetivo de este examen es **evaluar la correcta aplicación de los conceptos y técnicas** vistos hasta el momento:

- Correcta implementación de constructores.
- Modularización reutilizable y mantenible con uso de métodos con correcta parametrización y correcto encapsulamiento, publicando *setters* y *getters* sólo cuando corresponda.
- Manejo de clases, enumerados y colecciones.

Enunciado

Una empresa tiene que organizar la distribución de paquetes desde la casa matriz hacia sus sucursales utilizando una flota propia de fletes. El objetivo es distribuir los paquetes en forma rápida y eficiente, para lo cual se comenzó a desarrollar un programa. El programador asignado dejó la tarea inconclusa y nuestra misión es completarla.
De cada paquete conocemos la sucursal a la cual hay que enviarlo y su peso. Al registrarlo el sistema le asigna un número único de seguimiento. Esto ya está desarrollado.
El peso de cada paquete es importante porque cada flete tiene una capacidad máxima de carga, la cual no puede ser superada y además cada flete agrupa paquetes del mismo destino. Los fletes no tienen un destino asignado de antemano, pero este queda determinado por el destino del primer paquete que recibe. A partir de ahí solo aceptan paquetes con el mismo destino hasta llegar al 80% de su capacidad. Cuando se llega a ese umbral el flete se despacha, siendo eliminado de la lista de fletes disponibles y dándolo de alta en la lista de fletes despachados.
Los paquetes primero se juntan en un cuarto. A estos se los conoce como paquetes pendientes. Nuestra tarea principal es desarrollar un programa que optimice la distribución de estos paquetes entre los fletes disponibles, agrupándolos por destino sin sobrepasar la carga máxima. Por una cuentiòn de calidad de servicio, si durante este proceso un flete supera el 80% de carga, se lo considera despachable y se despacha, sin importar si todavía queda lugar para llevar otros paquetes.
Debido a ciertas quejas sobre el peso máximo de los paquetes, nos solicitan desarrollar un listado de fletes despachados indicando la patente del vehículo, su destino y los datos de su paquete más pesado (con número de seguimiento y peso).
Por algún problema con el código fuente la clase Flete está incompleta, aunque estos métodos están bien documentados en el código. Tenemos que completarlos.
Por último, un tester nos reportó que, aunque parece que el alta de fletes funciona bien, es posible dar de alta el mismo flete más de una vez, lo cual no sería correcto. El tester incorporó al final un caso más a la clase de Test y nos solicitó buscar y corregir el problema.

En resumen (y en orden de resolución):

1. Completar la clase **Flete** en base a la documentación del código fuente
    a. public Flete(String patente, double cargaMaxima)
    b. public boolean mismaPatente(String patente)
    c. public boolean esDespachable()

    d. private boolean puedeCargar(Paquete paquete)
    e. public boolean cargarPaquete(Paquete paquete)

2. Desarrollar el método **despacharPendientes()** de la clase **CentralDeDespachos** que procese todos paquetes pendientes y en el proceso, despachando los fletes que corresponda.

3. Desarrollar el método **listarFletesDespachadosConPaqueteMasPesado()** de la clase **CentralDeDespachos** que liste los fletes despachados, cada uno con su paquete más pesado.

4. Corregir el **bug** que permite registrar un flete duplicado si el mismo ya ha sido despachado.

5. Implementar al menos 2 interfaces.

6. Implementar Generics <T extends Object> | <? extends Object>.

7. Correcto uso y manejo de excepciones (pueden o no ser customizadas).

8. Crear la correcta arquitectura de paquetes.

Importante:

- Estudiar el código fuente y reusar los métodos ya desarrollados.
- Aprovechar el código de la clase Test y la salida esperada.
- Modularizar correctamente, desarrollando la funcionalidad en la clase que corresponda.
- No permitir que una clase pueda modificar en forma directa los atributos de otra clase.

Para la clase Test provista, la salida esperada es la siguiente:

```
Agregamos algunos fletes:
agregarFlete(AA111CD, 100.0): true
agregarFlete(AB222EF, 120.0): true
agregarFlete(AC333GH, 100.0): true
agregarFlete(AD444JK, 120.0): true
agregarFlete(AC333GH, 100.0): false

Agregamos algunos paquetes para despachar:
agregarPaquete(ALMAGRO, 30.0)
agregarPaquete(PALERMO, 45.0)
agregarPaquete(BOEDO, 20.0)
agregarPaquete(FLORES, 35.0)
agregarPaquete(ALMAGRO, 40.0)
agregarPaquete(PALERMO, 25.0)
agregarPaquete(BOEDO, 15.0)
agregarPaquete(FLORES, 20.0)
agregarPaquete(ALMAGRO, 30.0)
agregarPaquete(PALERMO, 35.0)
agregarPaquete(BOEDO, 20.0)
agregarPaquete(FLORES, 40.0)
agregarPaquete(ALMAGRO, 35.0)
agregarPaquete(PALERMO, 25.0)
agregarPaquete(BOEDO, 20.0)
agregarPaquete(FLORES, 15.0)
agregarPaquete(RETIRO, 50.0)

Despachamos los paquetes pendientes:
Paquetes despachados: 14
Paquetes pendientes: 3

No debería permitir agregar este flete:
agregarFlete(AB222EF, 120.0): false

Listado de fletes despachados con su paquete más pesado:


Patente: AD444JK Destino: FLORES Paquete: 12 Peso: 40.0
```