```java
package main;
import java.util.ArrayList;
import java.util.List;
class Sucursal {
  String nombre;
  int monto; // Positivo para cobro, negativo para pago
  public Sucursal(String nombre, int monto) {
    this.nombre = nombre;
    this.monto = monto;
  }
  @Override
  public String toString() {
    return "Sucursal{" +
        "nombre='" + nombre + '\'' +
        ", monto=" + monto +
        '}';
  }
}
public class Actividad3 {
  public static List<String> planificarRecorrido(List<Sucursal> sucursales, int saldoInicial) {
    List<String> recorrido = new ArrayList<>();
    int saldoActual = saldoInicial;
    List<Sucursal> sucursalesVisitadas = new ArrayList<>();
    //Mientras haya sucursales sin visitar
    while (sucursalesVisitadas.size() < sucursales.size()) {
      Sucursal mejorOpcion = null;
      int maxGanancia = Integer.MIN_VALUE; // Inicializar con el peor valor posible
      // Encontrar la mejor opción (mayor ganancia y factible) entre las no visitadas.
      for (Sucursal sucursal : sucursales) {
        //Si la sucursal ya fue visitada, continuar
        if (sucursalesVisitadas.contains(sucursal)) {
          continue;
        }
        // Si hay saldo suficiente, o si la sucursal implica un cobro.
        if (saldoActual + sucursal.monto >= 0) {
          //Priorizar los cobros. Si es un cobro, se le asigna un valor mas grande
          if (sucursal.monto > 0 && sucursal.monto > maxGanancia) {
              //Se encontro una mejor opcion de cobro
              maxGanancia = sucursal.monto;
              mejorOpcion = sucursal;
          } else if(mejorOpcion == null || mejorOpcion.monto < 0){ //Si no se encontro opcion de
cobro, elegir la que menos haga perder
            if (sucursal.monto > maxGanancia) {
              maxGanancia = sucursal.monto; //Es un pago, pero el menos negativo
              mejorOpcion = sucursal;
            }
          }
        }
      }
      // Si no se encontro una opcion viable, se termina
      if (mejorOpcion == null) {
        System.out.println("No se pudo completar el recorrido. Saldo insuficiente.");
```

```
            return recorrido;
        }
        // Se encontro una opcion
        saldoActual += mejorOpcion.monto;
        recorrido.add(mejorOpcion.nombre);
        sucursalesVisitadas.add(mejorOpcion); // Marcar la sucursal como visitada.
    }
    return recorrido;
  }
  public static void main(String[] args) {
    List<Sucursal> sucursales = new ArrayList<>();
    sucursales.add(new Sucursal("Sucursal A", 500));
    sucursales.add(new Sucursal("Sucursal B", -200));
    sucursales.add(new Sucursal("Sucursal C", 300));
    sucursales.add(new Sucursal("Sucursal D", -100));
    sucursales.add(new Sucursal("Sucursal E", 400));
    sucursales.add(new Sucursal("Sucursal F", -300));
    sucursales.add(new Sucursal("Sucursal G", 600));
    sucursales.add(new Sucursal("Sucursal H", -150));
    int saldoInicial = 50; // Ejemplo con saldo inicial
    List<String> recorrido = planificarRecorrido(sucursales, saldoInicial);
    if (!recorrido.isEmpty()) {
        System.out.println("Recorrido planificado:");
        System.out.println(recorrido);
    }

  }
}
```

¿Cómo diseñarías un recorrido que minimice el riesgo de quedarse sin dinero?

El recorrido se puede estructurar de la siguiente forma:

1. Iniciar con un saldo inicial: El cobrador empieza con una cantidad de dinero determinada.
2. Priorizar las sucursales con mayor cobro: Primero se visitan las sucursales que ofrecen el cobro más alto, para asegurar un incremento en el saldo. Si el cobrador no tiene suficiente saldo para cubrir un pago, no se debe visitar esa sucursal.
3. Verificar el saldo antes de visitar una sucursal con pago: Si la siguiente sucursal implica un pago (valor negativo), el cobrador debe asegurarse de que su saldo sea suficiente para cubrir el pago antes de visitarla.
4. Visitar las sucursales de forma secuencial: Una vez se visitan todas las sucursales posibles sin que el saldo se agote, el recorrido se termina. Si en algún momento no hay suficiente saldo para continuar, se debe seleccionar la siguiente mejor opción, que sería la sucursal con el menor valor negativo o con el mayor cobro restante que permita seguir avanzando.

¿Qué heurística podrías aplicar para tomar decisiones sobre el orden de visita?

Heurística:

- Priorizar las sucursales con mayor cobro (monto positivo) en primer lugar.
- Si no hay más sucursales con cobro, se selecciona la sucursal con el menor pago negativo que pueda ser cubierto con el saldo disponible.

Este enfoque busca la mejor opción local en cada paso (es decir, seleccionar la sucursal con el mayor cobro posible) y asegura que el saldo nunca sea insuficiente para cubrir un pago.

Este enfoque no garantiza la solución óptima global, ya que es un enfoque greedy (se toma la mejor decisión en cada paso sin mirar el futuro), pero ayuda a minimizar el riesgo al maximizar los cobros en cada etapa del recorrido y minimizar las pérdidas con pagos.