**package** assignment1;  
  
  
**import** com.sun.org.apache.xpath.internal.operations.Bool;  
  
**import** java.util.\*;  
  
*/\*\*  
 \* A graph like structure to store all the warehouses and performs DFS  
 \*/***public class** WarehouseManager {  
  
 **private** Set<Warehouse> **warehouses** = **null**;  
  
 **public** WarehouseManager(Set<Warehouse> warehouses) {  
 **this**.**warehouses** = warehouses;  
 }  
  
 **public** List<Transfer> BFS(Warehouse start) {  
 *// Map <child -> parent>* Map<Warehouse, Edge> cameFrom = **new** HashMap<>();  
 *// LinkedList as a Queue* List<Warehouse> queueForWarehouse = **new** LinkedList<>();  
 Set<ItemType> visitedType = **new** HashSet<>();  
 Set<Warehouse> unvisitedWarehouse = **new** HashSet<>(**this**.**warehouses**); *// available warehouses* Warehouse current; *// keep track on current visiting warehouse* Warehouse destination = **null**; *// you destination which is the first warehouse making transactions  
 // valid* Set<Warehouse> greyWarehouse = **new** HashSet<>();  
  
 queueForWarehouse.add(start);  
 cameFrom.put(start, **null**);  
  
 **while** (destination == **null** && !queueForWarehouse.isEmpty()) {  
 current = queueForWarehouse.remove(0);  
 unvisitedWarehouse.remove(current);  
 *// mark current as visited, so it will not be considered as a neighbour* Set<ItemType> possibleTypes = current.availableItemTypes();  
 *// only available items are possible items to make transfers* possibleTypes.removeAll(visitedType);  
 *// visited types will not be visited no more* List<Edge> neighbours = getNeighbours(possibleTypes, unvisitedWarehouse);  
 *//System.out.println(neighbours.size());  
 // grey warehouses are candidates and make sure you don't enqueue the same warehouse twice* **for** (Edge e : neighbours) {  
 **if** (greyWarehouse.add(e.getWarehouse())) {  
 *// enqueuing and checking if the transfer is valid* queueForWarehouse.add(e.getWarehouse());  
 }  
 visitedType.add(e.getType());  
 cameFrom.put(e.getWarehouse(), **new** Edge(current, e.getType()));  
 **if** (e.getWarehouse().canAddItem(e.getType())) {  
 *// found destination* destination = e.getWarehouse();  
 **break**;  
 }  
 }  
 }  
  
 List<Transfer> transfers = **new** ArrayList<>();  
 current = destination;  
 **while** (cameFrom.get(current) != **null**) {  
 transfers.add(**new** Transfer(cameFrom.get(current).getWarehouse(), current,  
 cameFrom.get(current).getType()));  
 current = cameFrom.get(current).getWarehouse();  
 }  
  
 **if**(transfers.size() == 0) {  
 *// no possible transfer* **return null**;  
 } **else** {  
 *//System.out.println(transfers);* **return** transfers;  
 }  
 }  
  
  
 */\*\*  
 \* Get transfer candidate edges  
 \* candidates are with possible Item types and unvisited warehouses  
 \** ***@param types*** *the types can be transfer from source  
 \** ***@return*** *valid possible transfer destinations via type  
 \*/* **private** List<Edge> getNeighbours(Set<ItemType> types, Set<Warehouse> unvisitedWarehouse) {  
 List<Edge> neighbours = **new** LinkedList<>();  
 **for** (ItemType type : types) {  
 **for** (Warehouse w : unvisitedWarehouse) {  
 **if** (w.getValidTypes().contains(type)) {  
 neighbours.add(**new** Edge(w, type));  
 }  
 }  
 }  
 **return** neighbours;  
 }  
  
 @Override  
 **public** String toString() {  
 String output = **""**;  
 **for**(Warehouse w : **this**.**warehouses**) {  
 output += w;  
 output += **"\n"**;  
 }  
 **return** output;  
 }  
}