Violation of 3 SOLID Principles

The below code violates:

(1) Single-responsibility principle, since the reestablishObservers, the account operations and the client operations of the bank should not be all the reason for change in the same class. They should be divided in 3 modules: a wrapper for the bank class to handle the observer side, a module for client operations and a module for account operations.

(1) Interface-segregation principle since some methods that operate on clients and methods that operate on accounts should be divided in different modules.

(2) Open-closed principle, because the Bank class should implement an interface which would give it a well-defined, stable description.

package model;  
  
import utils.AccountFactory;  
import utils.AccountType;  
  
import java.io.Serializable;  
import java.util.HashMap;  
import java.util.LinkedList;  
  
*/\*\*  
 \* Created by Ioan on 08.05.2017.  
 \*/*public class Bank implements Serializable {  
  
 private HashMap<Person, LinkedList<Account>> personAccountMap;  
 private AccountFactory accountFactory;  
  
 public Bank() {  
 this.personAccountMap = new HashMap<Person, LinkedList<Account>>();  
 this.accountFactory = new AccountFactory(0,5);  
 }  
  
 public HashMap<Person, LinkedList<Account>> getPersonAccountMap() {  
 return personAccountMap;  
 }  
  
 private boolean isWellFormed() {  
 LinkedList<Person> people = new LinkedList<>();  
 people.addAll(personAccountMap.keySet());  
 for (int i = 0; i < people.size() - 1; i++) {  
 for (int j = i + 1; j < people.size(); j++) {  
 if (people.get(i).equals(people.get(j)))  
 return false;  
 }  
 }  
 return true;  
 }  
  
 public void reestablishObservers() {  
 for (Person p: personAccountMap.keySet()) {  
 LinkedList<Account> accounts = personAccountMap.get(p);  
 for (Account account: accounts) {  
 account.addObserver(p);  
 }  
 }  
 }  
  
   
 public Person addPerson(int id, String email, String name, String telephone, double balance, int accountId, AccountType accountType) {  
 assert email.compareTo("") != 0;  
 assert telephone.compareTo("") != 0;  
 assert name.compareTo("") != 0;  
 assert balance >= 0;  
 assert id >= 0;  
 assert accountId >= 0;  
 Person person = new Person(name, id, telephone, email);  
 Account account = accountFactory.getAccount(accountType);  
 account.deposit(balance);  
 account.setId(accountId);  
 account.addObserver(person);  
 LinkedList<Account> accounts = new LinkedList<>();  
 accounts.add(account);  
 if (personAccountMap.containsKey(person)) {  
 assert isWellFormed();  
 return null;  
 } else {  
 personAccountMap.put(person, accounts);  
 assert isWellFormed();  
 return person;  
 }  
 }  
  
   
 public Person removePerson(int id) {  
 assert id >= 0;  
 Person person = getPerson(id);  
 if (person == null) {  
 assert isWellFormed();  
 return null;  
 } else {  
 personAccountMap.remove(person);  
 assert isWellFormed();  
 return person;  
 }  
 }  
  
   
 public Person getPerson(int id) {  
 assert id >= 0;  
 for (Person p : personAccountMap.keySet()) {  
 if (p.getId() == id) {  
 assert isWellFormed();  
 return p;  
 }  
 }  
 assert isWellFormed();  
 return null;  
 }  
  
   
 public Person editPerson(int id, String email, String name, String telephone) {  
 assert email.compareTo("") != 0;  
 assert telephone.compareTo("") != 0;  
 assert name.compareTo("") != 0;  
 assert id >= 0;  
 Person p = getPerson(id);  
 if (p != null) {  
 LinkedList<Account> accounts = personAccountMap.get(p);  
 p.setEmail(email);  
 p.setName(name);  
 p.setTelephone(telephone);  
 removePerson(id);  
 personAccountMap.put(p, accounts);  
 assert isWellFormed();  
  
 return p;  
 } else {  
 assert isWellFormed();  
 return null;  
 }  
 }  
  
   
 public LinkedList<Account> getAccounts(int id) throws NullPointerException {  
  
 assert id >= 0;  
 assert isWellFormed();  
 return personAccountMap.get(getPerson(id));  
  
 }  
  
   
 public Person addAccount(int personId, double balance, int accountId, AccountType accountType) {  
 assert balance >= 0;  
 assert personId >= 0;  
 assert accountId >= 0;  
 Account account = accountFactory.getAccount(accountType);  
 account.deposit(balance);  
 account.setId(accountId);  
 Person person = getPerson(personId);  
 account.addObserver(person);  
 if (person != null)  
 if (getAccounts(personId).contains(account)) {  
 assert isWellFormed();  
 return null;  
 } else {  
 getAccounts(personId).add(account);  
 }  
 assert isWellFormed();  
 return person;  
 }  
  
   
 public Account removeAccount(int personId, int accountId) {  
 assert personId >= 0;  
 assert accountId >= 0;  
 Person person = getPerson(personId);  
 if (person != null) {  
 Account account = getAccount(personId, accountId);  
 if (account != null) {  
 getAccounts(personId).remove(account);  
 assert isWellFormed();  
 return account;  
 } else {  
 assert isWellFormed();  
 return null;  
 }  
 } else {  
 assert isWellFormed();  
 return null;  
 }  
 }  
  
   
 public Account getAccount(int personId, int accountId) {  
 assert personId >= 0;  
 assert accountId >= 0;  
 for (Account account : getAccounts(personId)) {  
 if (account.getId() == accountId) {  
 assert isWellFormed();  
 return account;  
 }  
 }  
 assert isWellFormed();  
 return null;  
 }  
  
   
 public double deposit(int personId, int accountId, double sum) {  
 assert personId >= 0;  
 assert accountId >= 0;  
 assert sum >= 0;  
 for (Account account : getAccounts(personId)) {  
 if (account.getId() == accountId) {  
 assert isWellFormed();  
 return account.deposit(sum);  
 }  
 }  
 assert isWellFormed();  
 return -1;  
 }  
  
   
 public double withdraw(int personId, int accountId, double sum) {  
 assert personId >= 0;  
 assert accountId >= 0;  
 assert sum >= 0;  
 for (Account account : getAccounts(personId)) {  
 if (account.getId() == accountId) {  
 assert isWellFormed();  
 return account.withdraw(sum);  
 }  
 }  
 assert isWellFormed();  
 return -1;  
 }  
  
  
  
}