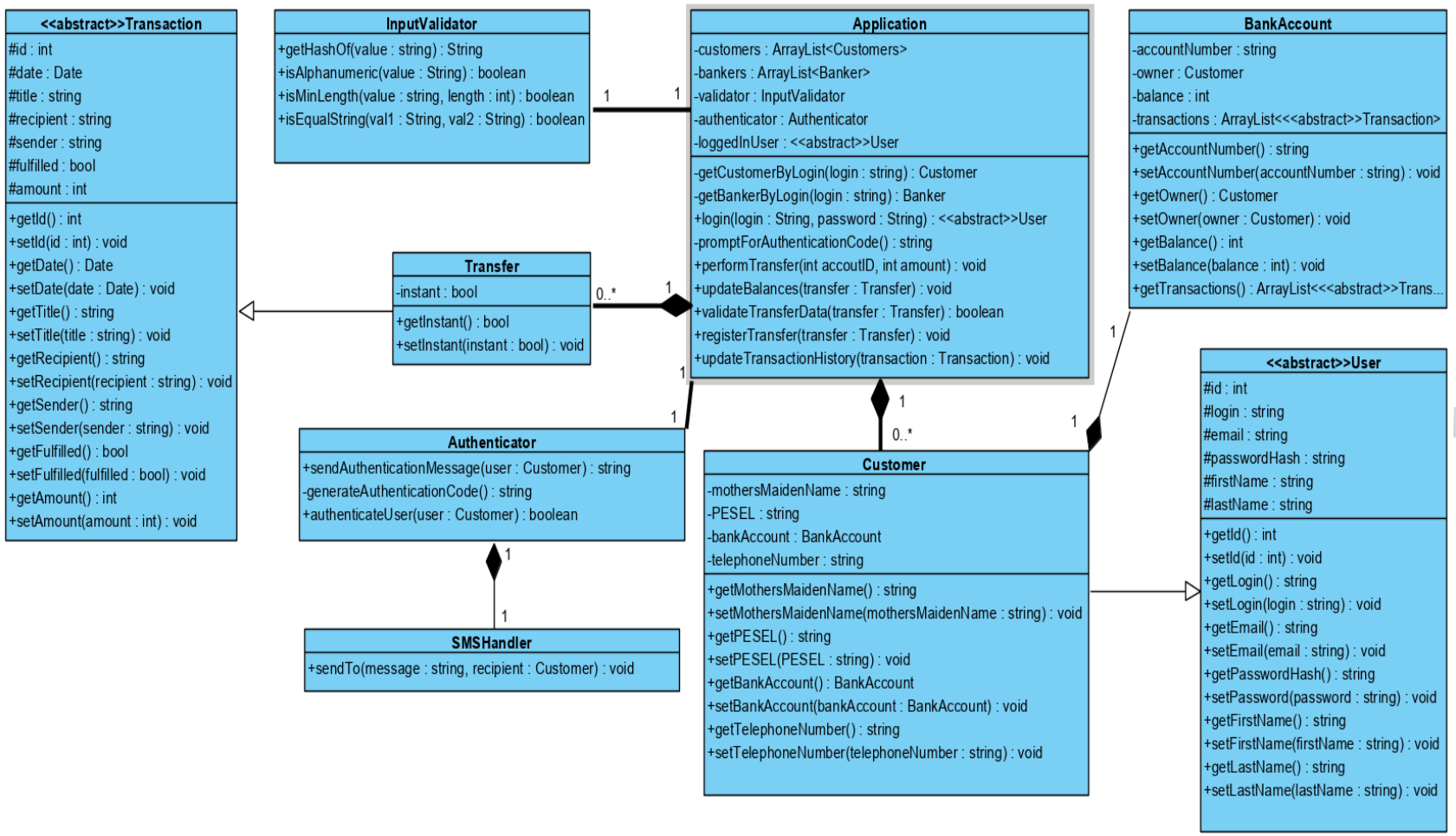
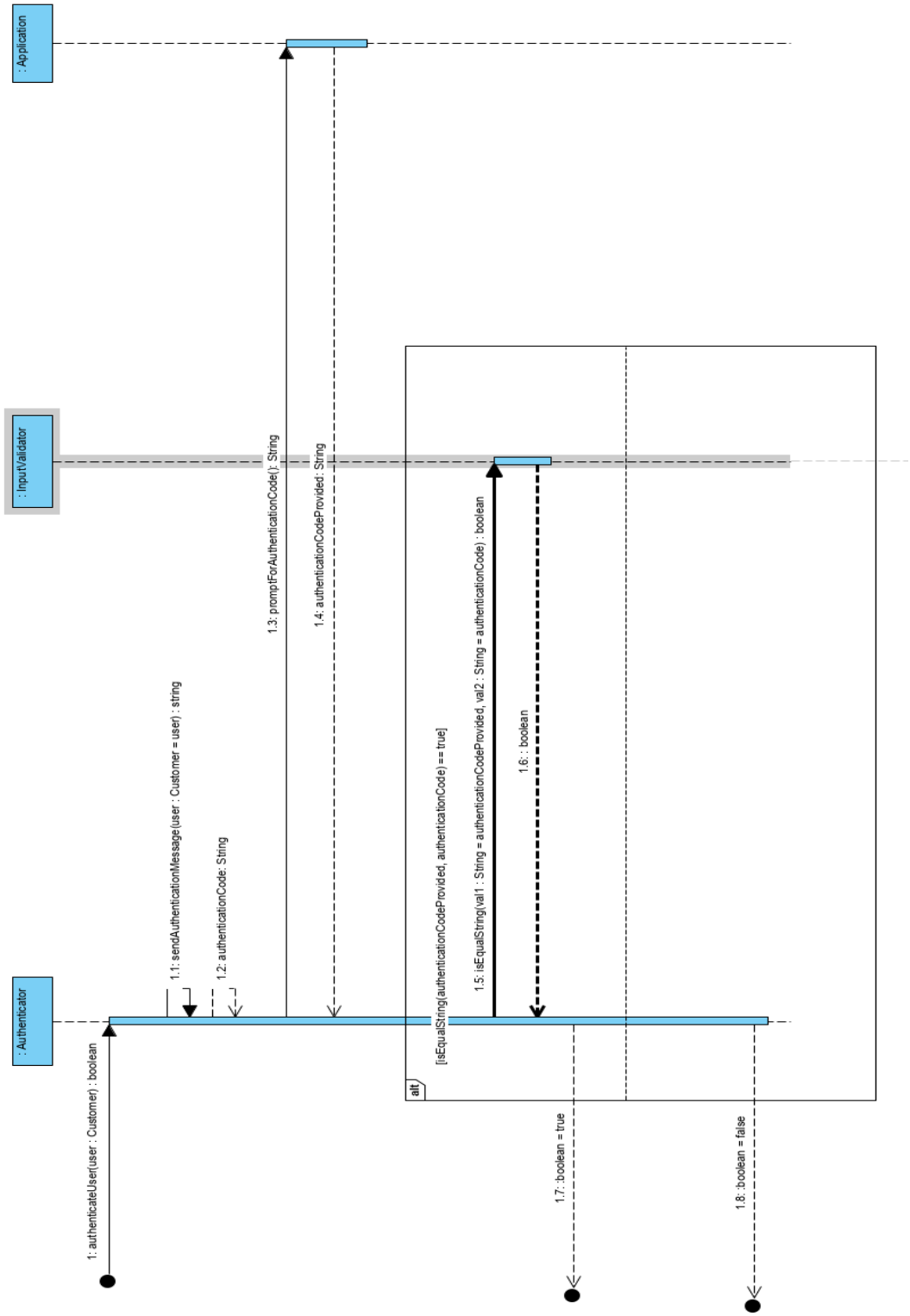
Inżynieria Oprogramowania
Sprawozdanie

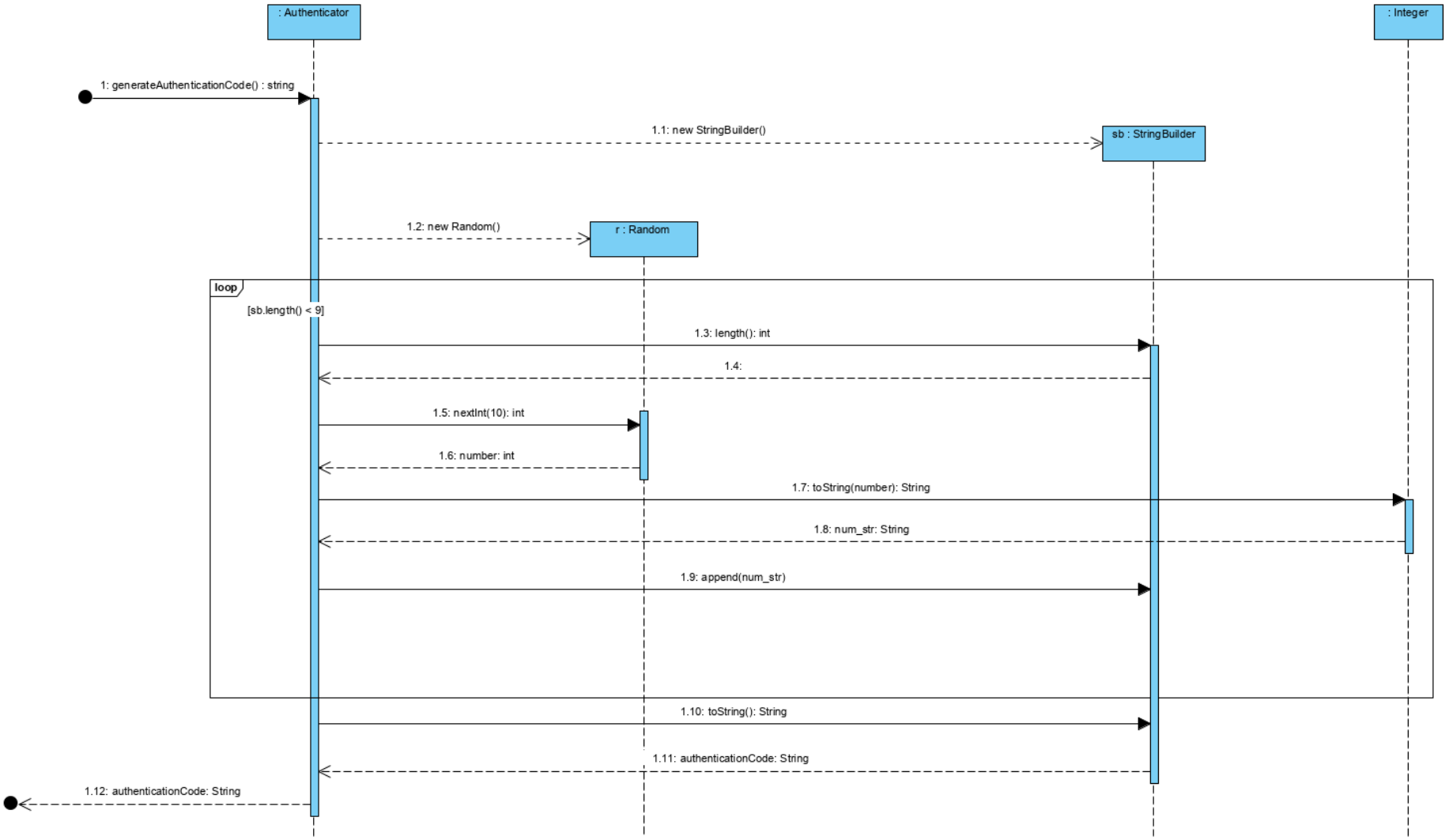
Daniel Król
Artur Boryś

Kod całego projektu znajduje się pod linkiem: https://github.com/SheCanWait/IO

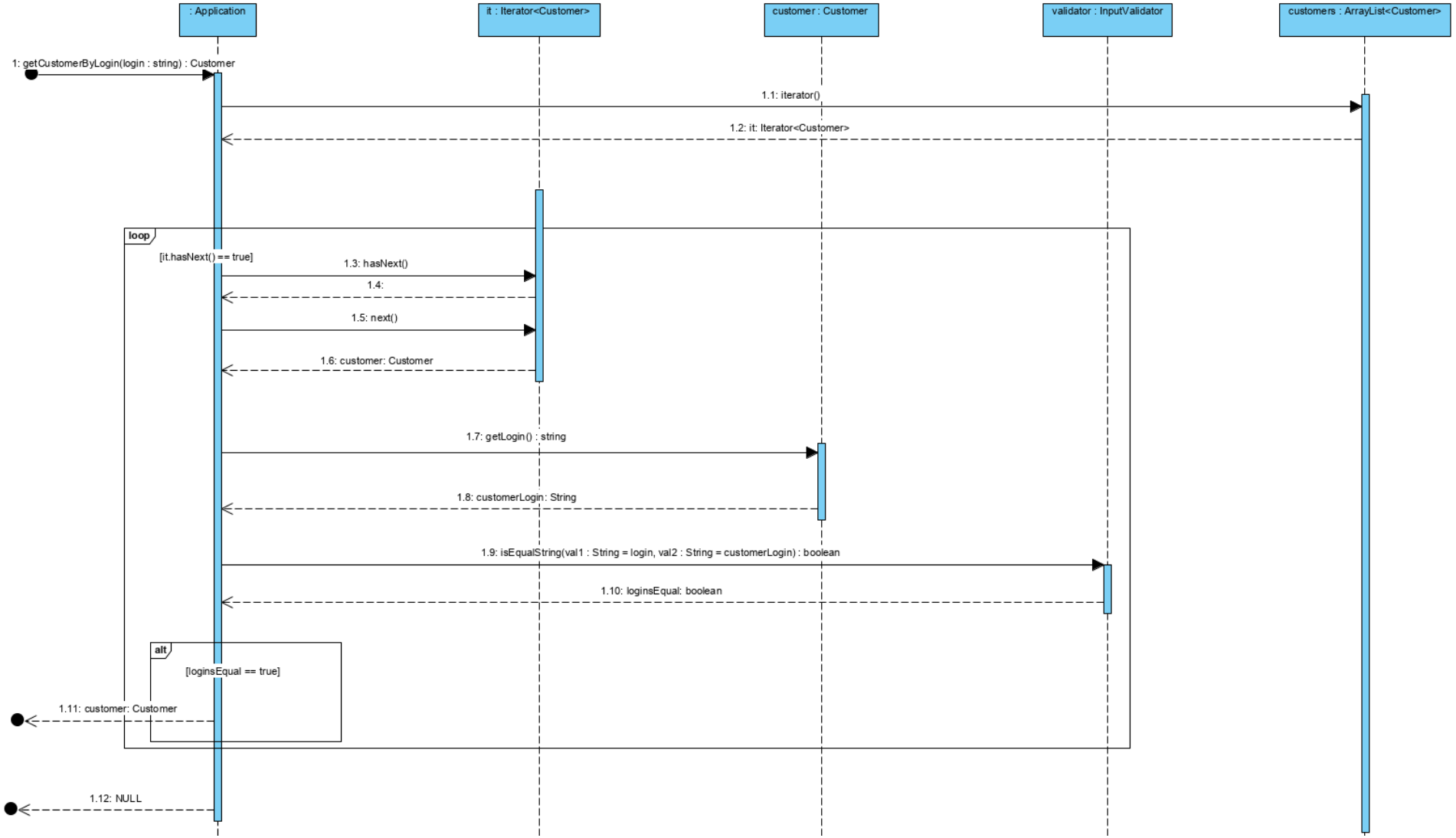

<<abstract>>Transaction
#id : int
#date : Date
#title : string
#recipient : string
#sender : string
#fulfilled : bool
#amount : int
+getId() : int
+setId(id : int) : void
+getDate() : Date
+setDate(date : Date) : void
+getTitle() : string
+setTitle(title : string) : void
+getRecipient() : string
+setRecipient(recipient : string) : void
+getSender() : string
+setSender(sender : string) : void
+getFulfilled() : bool
+setFulfilled(fulfilled : bool) : void
+getAmount() : int
+setAmount(amount : int) : void
InputValidator
+getHashOf(value : string) : String
+isAlphanumeric(value : String) : boolean
+isMinLength(value : string, length : int) : boolean
+isEqualString(val1 : String, val2 : String) : boolean
Application
-customers : ArrayList<Customers>
-bankers : ArrayList<Banker>
-validator : InputValidator
-authenticator : Authenticator
-loggedInUser : <<abstract>>User
-getCustomerByLogin(login : string) : Customer
-getBankerByLogin(login : string) : Banker
+login(login : String, password : String) : <<abstract>>User
-promptForAuthenticationCode() : string
+performTransfer(int accoutID, int amount) : void
+updateBalances(transfer : Transfer) : void
+validateTransferData(transfer : Transfer) : boolean
+registerTransfer(transfer : Transfer) : void
+updateTransactionHistory(transaction : Transaction) : void
BankAccount
-accountNumber : string
-owner : Customer
-balance : int
-transactions : ArrayList<<<abstract>>Transaction>
+getAccountNumber() : string
+setAccountNumber(accountNumber : string) : void
+getOwner() : Customer
+setOwner(owner : Customer) : void
+getBalance() : int
+setBalance(balance : int) : void
+getTransactions() : ArrayList<<<abstract>>Trans...
Transfer
-instant : bool
+getInstant() : bool
+setInstant(instant : bool) : void
Authenticator
+sendAuthenticationMessage(user : Customer) : string
-generateAuthenticationCode() : string
+authenticateUser(user : Customer) : boolean
SMSHandler
+sendTo(message : string, recipient : Customer) : void
Customer
-mothersMaidenName : string
-PESEL : string
-bankAccount : BankAccount
-telephoneNumber : string
+getMothersMaidenName() : string
+setMothersMaidenName(mothersMaidenName : string) : void
+getPESEL() : string
+setPESEL(PESEL : string) : void
+getBankAccount() : BankAccount
+setBankAccount(bankAccount : BankAccount) : void
+getTelephoneNumber() : string
+setTelephoneNumber(telephoneNumber : string) : void
<<abstract>>User
#id : int
#login : string
#email : string
#passwordHash : string
#firstName : string
#lastName : string
+getId() : int
+setId(id : int) : void
+getLogin() : string
+setLogin(login : string) : void
+getEmail() : string
+setEmail(email : string) : void
+getPasswordHash() : string
+setPassword(password : string) : void
+getFirstName() : string
+setFirstName(firstName : string) : void
+getLastName() : string
+setLastName(lastName : string) : void
1
1
1
1
0..*
1
1
1
1
1
1
0..*
1
1
1



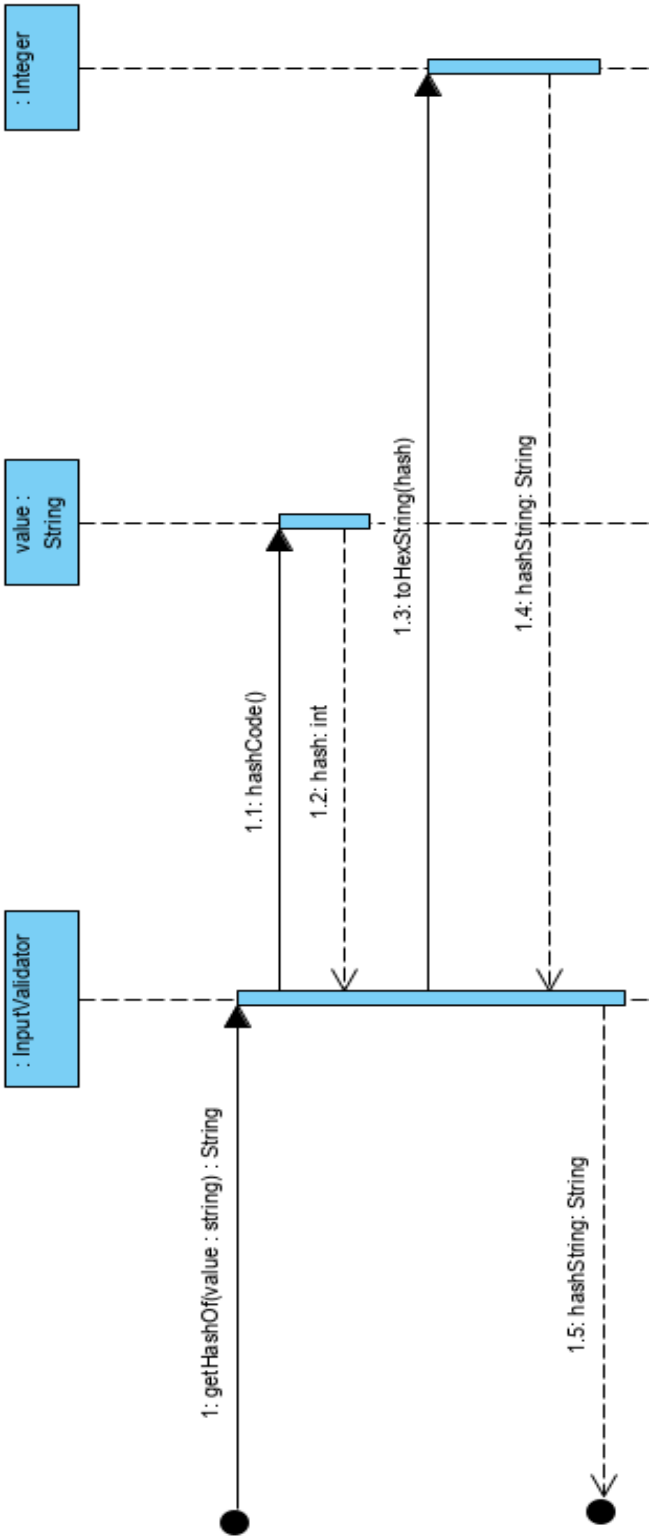

```
public static boolean authenticateUser(Customer user) {
    String authenticationCode = sendAuthenticationMessage(user);
    String authenticationCodeProvided = Application.promptForAuthenticationCode();
    return InputValidator.isEqualString(authenticationCodeProvided, authenticationCode);
  }
```


: Authenticator
: Integer
1: generateAuthenticationCode() : string
1.1: new StringBuilder()
sb : StringBuilder
1.2: new Random()
r : Random
loop
[sb.length() < 9]
1.3: length(): int
1.4:
1.5: nextInt(10): int
1.6: number: int
1.7: toString(number): String
1.8: num_str: String
1.9: append(num_str)
1.10: toString(): String
1.11: authenticationCode: String
1.12: authenticationCode: String

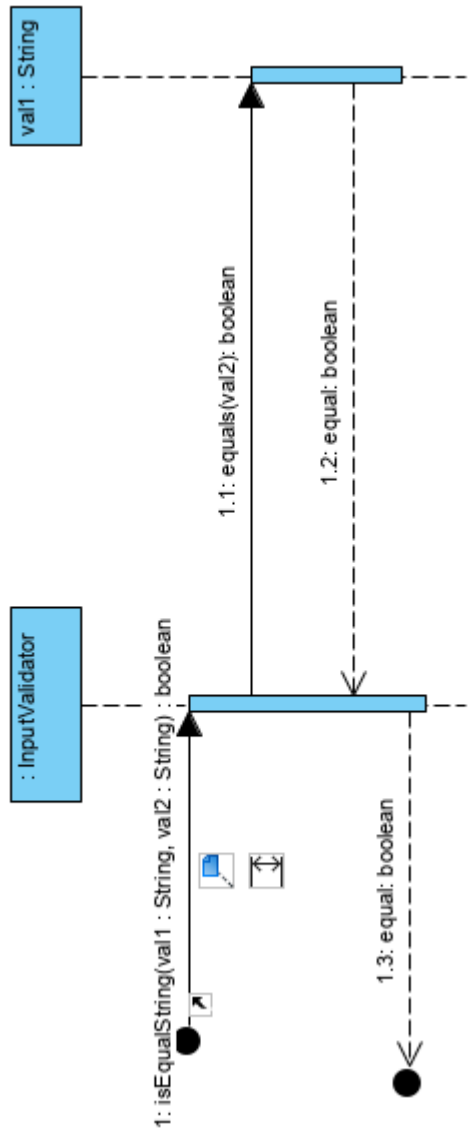
```
private static String generateAuthenticationCode() {
    StringBuilder sb = new StringBuilder();
    Random r = new Random();
    while(sb.length() < 9) {
       int number = r.nextInt(10);
       String num_str = Integer.toString(number);
       sb.append(num_str);
    }
    String authenticationCode = sb.toString();
    // Potrzebne do zasymulowania wpisywania kodu weryfikacyjnego przez użytkownika
    Application.lastAuthenticationCode = authenticationCode;
    return authenticationCode;
 }
```


: Application
it : Iterator<Customer>
customer : Customer
validator : InputValidator
customers : ArrayList<Customer>
1: getCustomerByLogin(login : string) : Customer
1.1: iterator()
1.2: it: Iterator<Customer>
loop
[it.hasNext() == true]
1.3: hasNext()
1.4:
1.5: next()
1.6: customer: Customer
1.7: getLogin() : string
1.8: customerLogin: String
1.9: isEqualString(val1 : String = login, val2 : String = customerLogin) : boolean
1.10: loginsEqual: boolean
alt
[loginsEqual == true]
1.11: customer: Customer
1.12: NULL

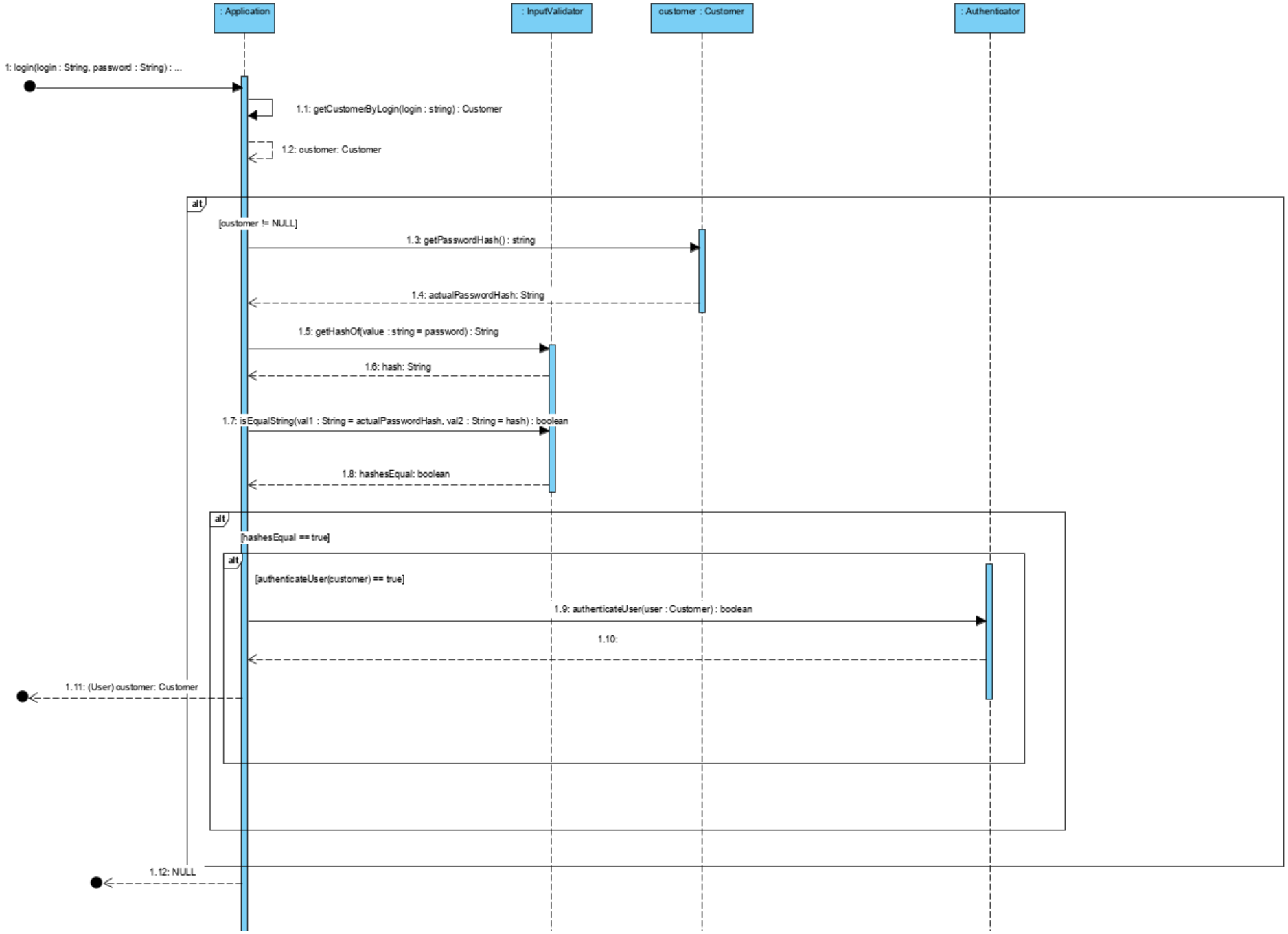
```
private Customer getCustomerByLogin(String login) {
   Iterator<Customer> it = customers.iterator();
   while(it.hasNext()) {
      Customer customer = it.next();
      String customerLogin = customer.getLogin();
      boolean loginsEqual = InputValidator.isEqualString(login, customerLogin);
      if(loginsEqual) {
         return customer;
      }
   }
   return null;
}
```



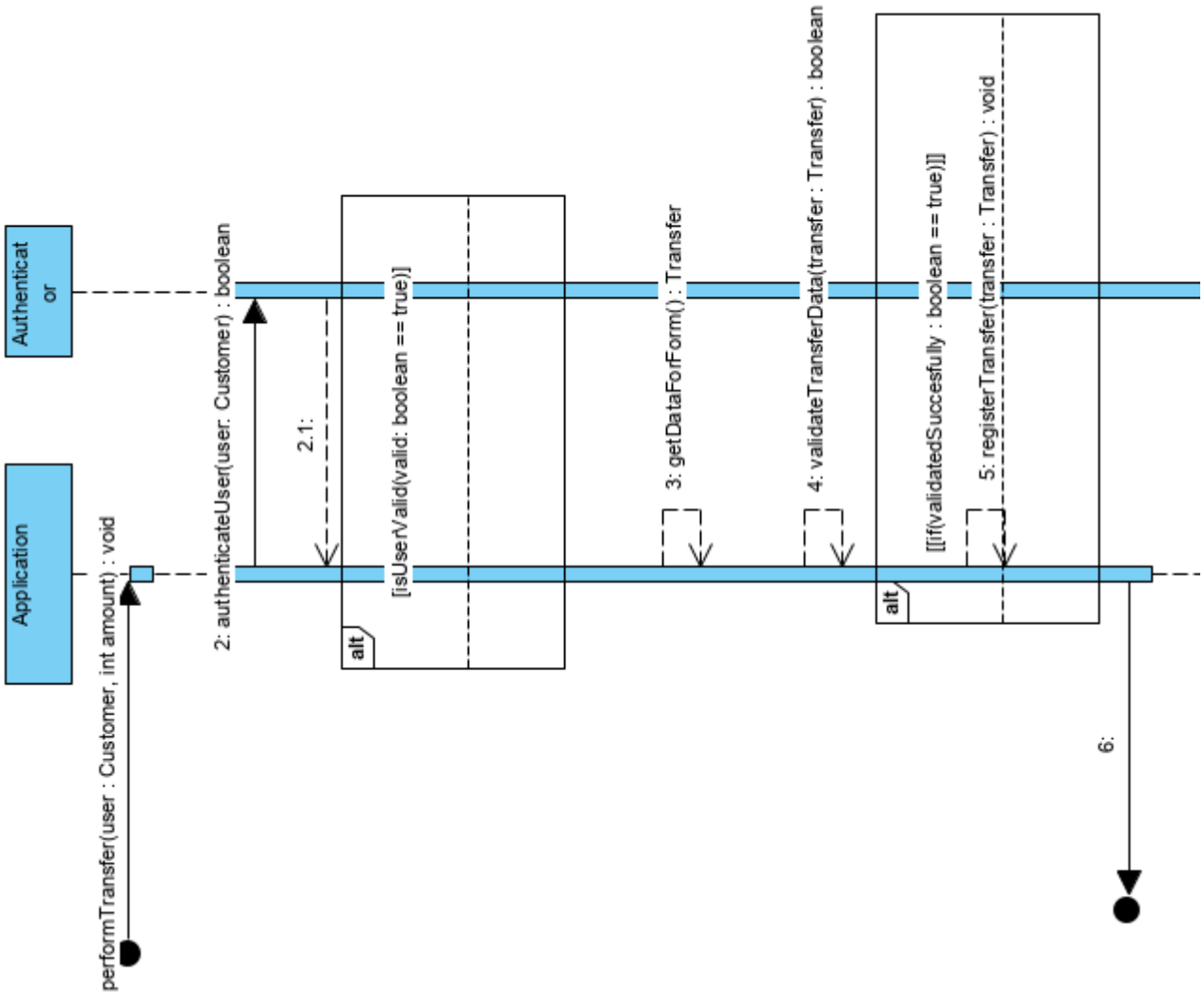

```
static String getHashOf(String value) {
   return Integer.toHexString(value.hashCode());
}
```




```
static boolean isEqualString(String val1, String val2) {
   boolean equal = val1.equals(val2);
   return equal;
}
```

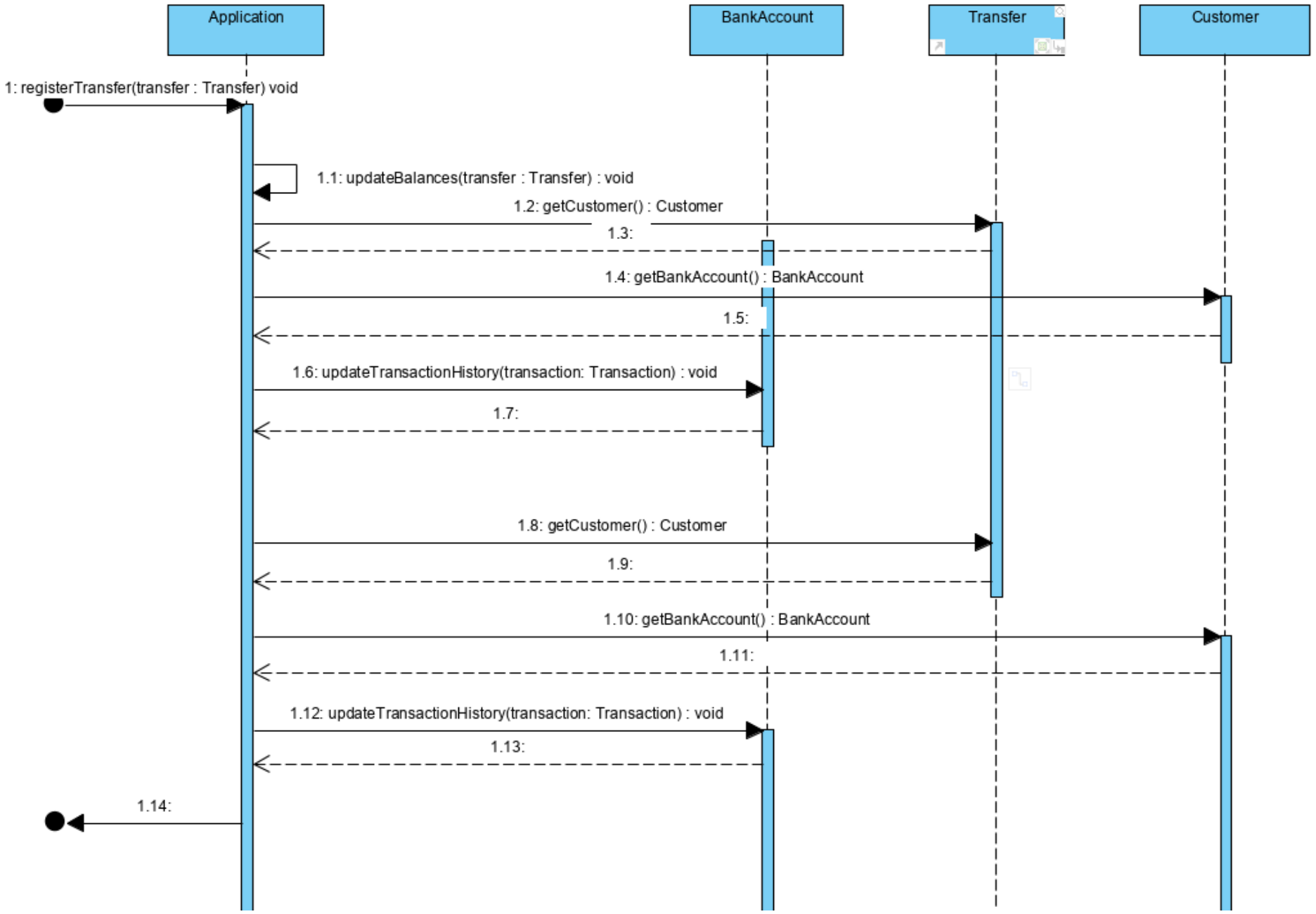

: Application
: InputValidator
customer : Customer
: Authenticator
1: login(login : String, password : String) : ...
1.1: getCustomerByLogin(login : string) : Customer
1.2: customer: Customer
alt
[customer != NULL]
1.3: getPasswordHash() : string
1.4: actualPasswordHash: String
1.5: getHashOf(value : string = password) : String
1.6: hash: String
1.7: isEqualString(val1 : String = actualPasswordHash, val2 : String = hash) : boolean
1.8: hashesEqual: boolean
alt
[hashesEqual == true]
alt
[authenticateUser(customer) == true]
1.9: authenticateUser(user : Customer) : boolean
1.10:
1.11: (User) customer: Customer
1.12: NULL

```
    public User login(String login, String password) {
        Customer customer = getCustomerByLogin(login);

        if(customer != null) {
            String actualPasswordHash = customer.getPasswordHash();
            String hash = InputValidator.getHashOf(password);
            boolean hashesEqual = InputValidator.isEqualString(actualPasswordHash, hash);
            if(hashesEqual) {
                if(Authenticator.authenticateUser(customer) == true) {
                    return customer;
                }
            }
        }

        return null;
    }
```

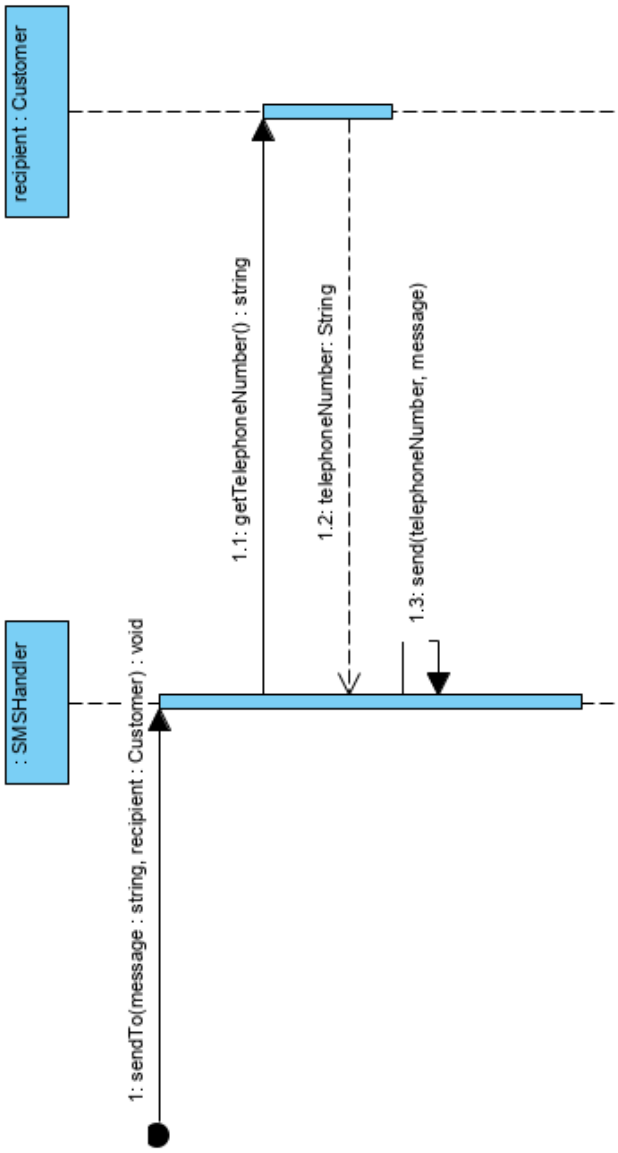



```
    public void performTransfer(Customer user, int amount) {
        if(Authenticator.authenticateUser(user) == true) {
            Transfer transfer = getDataForForm();
            if(validateTransferData(transfer) == true) {
```

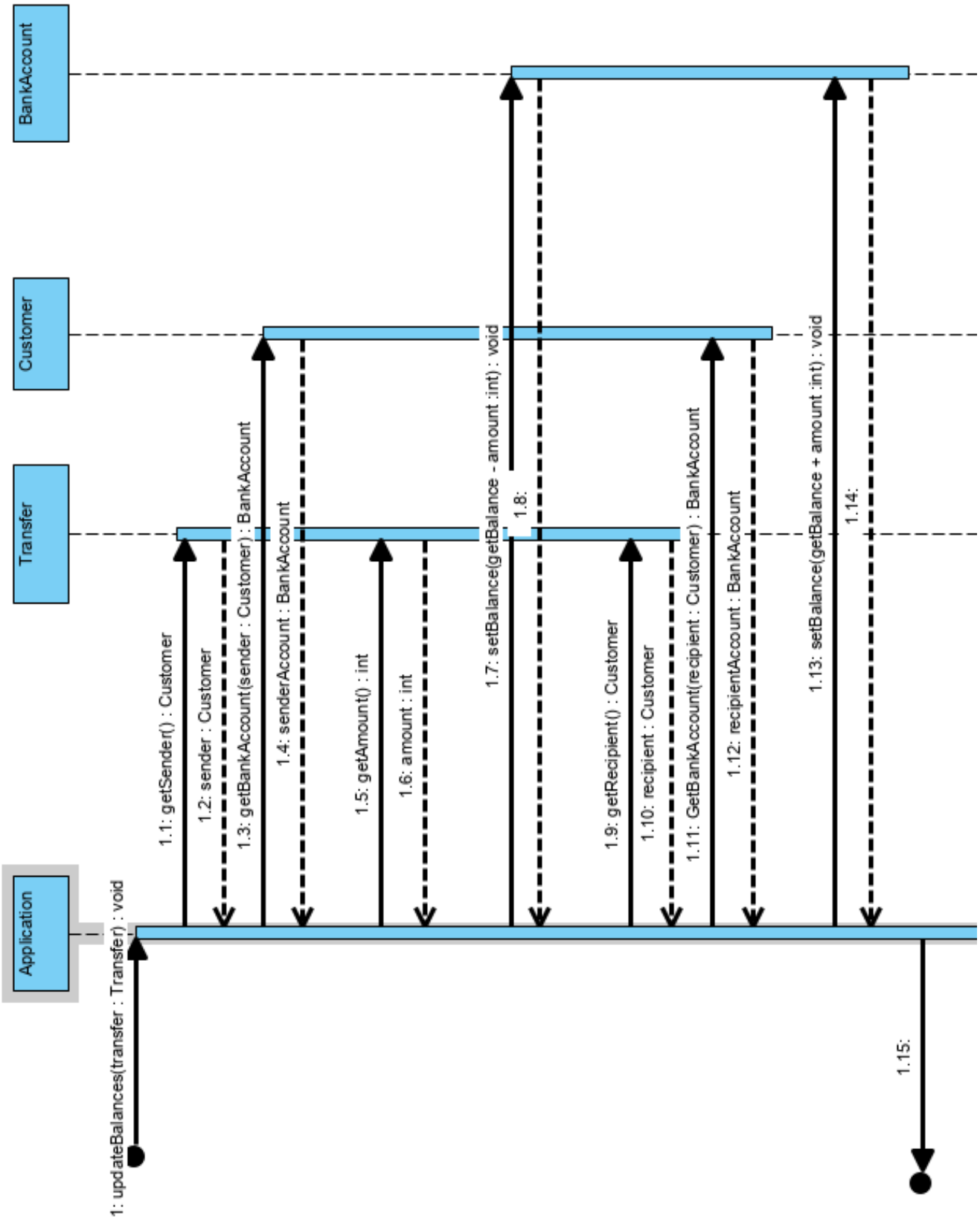
```
            registerTransfer(transfer);
        }
    }
}
```


Application
BankAccount
Transfer
Customer
1: registerTransfer(transfer : Transfer) void
1.1: updateBalances(transfer : Transfer) : void
1.2: getCustomer() : Customer
1.3:
1.4: getBankAccount() : BankAccount
1.5:
1.6: updateTransactionHistory(transaction: Transaction) : void
1.7:
1.8: getCustomer() : Customer
1.9:
1.10: getBankAccount() : BankAccount
1.11:
1.12: updateTransactionHistory(transaction: Transaction) : void
1.13:
1.14:

```
    private void registerTransfer(Transfer transfer) {
        updateBalances(transfer);
        transfer.recipient.getBankAccount().updateTransactionHistory(transfer);
        transfer.sender.getBankAccount().updateTransactionHistory(transfer);
}
```

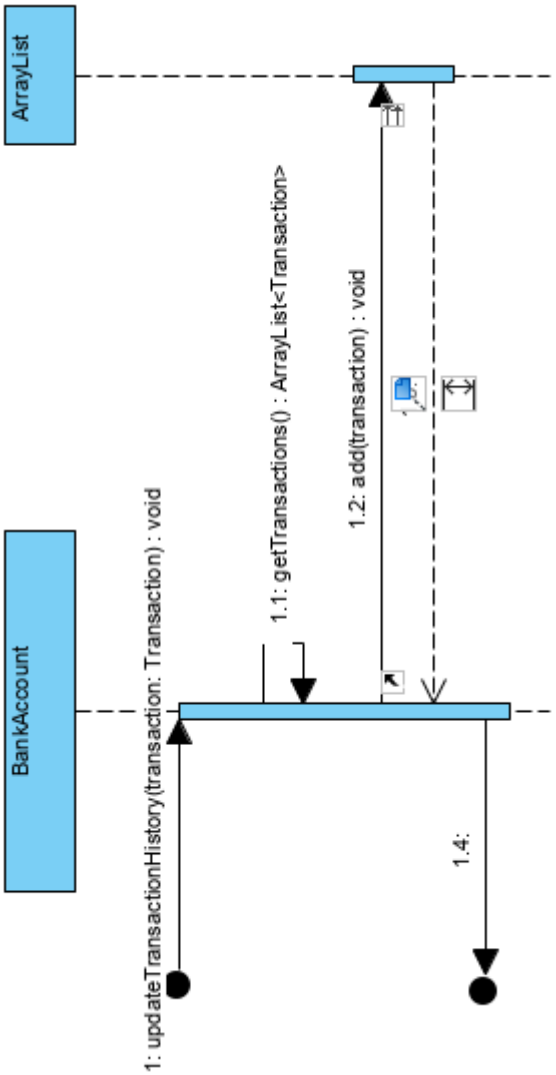



```
    private static String sendAuthenticationMessage(Customer user) {
        String authenticationCode = generateAuthenticationCode();
        StringBuilder sb = new StringBuilder();

        sb.append("Twoj kod weryfikacyjny: ");
        sb.append(authenticationCode);

        String message = sb.toString();

        SMSHandler.sendMessageTo(message, user);

        return authenticationCode;
}
```



private void updateBalances(Transfer transfer) {

```
        Customer sender = transfer.getSender();
        BankAccount senderAccount = sender.getBankAccount();
        senderAccount.setBalance(senderAccount.getBalance() - transfer.amount);

        Customer recipient = transfer.getSender();
        BankAccount recipientAccount = recipient.getBankAccount();
        recipientAccount.setBalance(recipientAccount.getBalance() + transfer.amount);
}
```




```
    public void updateTransactionHistory(Transfer transfer) {
        getTransactions().add(transfer);
}
```