1. **BankOperations.java**

```
public interface BankOperations {
    void deposit(double amount);
    void withdraw(double amount);
    void transfer(Account target, double amount);
    double checkBalance();
    void showTransactionHistory();
}
```

2.

Account.java

```
package oops_concept;
import java.util.ArrayList;
import java.util.List;


public abstract class Account {
      protected String accountNumber;
    protected double balance;
    protected List<String> transactionHistory;
    public Account(String accountNumber) {
        this.accountNumber = accountNumber;
        this.balance = 0.0;
        this.transactionHistory = new ArrayList<>();
    }
    public abstract void deposit(double amount);
    public abstract void withdraw(double amount);

    public void transfer(Account target, double amount) {
        if (amount <= 0) {
            addTransaction("Transfer failed: Invalid amount");
            throw new IllegalArgumentException("Amount must be positive");
        }
        if (target == null) {
            addTransaction("Transfer failed: Invalid target account");
            throw new IllegalArgumentException("Target account cannot be null");
        }
        if (this.balance < amount) {
            addTransaction("Transfer failed: Insufficient funds");
            throw new IllegalArgumentException("Insufficient funds for transfer");
        }

        this.withdraw(amount);
        target.deposit(amount);
        addTransaction("Transferred $" + amount + " to account " +
target.accountNumber);
    }

    public double checkBalance() {
        return balance;
    }

    protected void addTransaction(String info) {
        String transaction = java.time.LocalDateTime.now() + " - " + info;
        transactionHistory.add(transaction);
    }

    public void showTransactionHistory() {
        if (transactionHistory.isEmpty()) {
            System.out.println("No transactions found");
            return;
```

```
        }
        System.out.println("Transaction History for Account: " + accountNumber);
        for (String transaction : transactionHistory) {
            System.out.println(transaction);
        }
    }


}
```

3.

SavingsAccount.java

BankOperation.java

```
package banking;

public interface BankOperations {
    void deposit(double amount);
    void withdraw(double amount);
    void transfer(Account target, double amount);
    double checkBalance();
    void showTransactionHistory();
}
```

Account.java

```
package banking;

import java.util.ArrayList;
import java.util.List;

public abstract class Account {
    protected String accountNumber;
    protected double balance;
    protected List<String> transactionHistory;

    public Account(String accountNumber) {
        this.accountNumber = accountNumber;
        this.balance = 0.0;
        this.transactionHistory = new ArrayList<>();
    }

    public abstract void deposit(double amount);
    public abstract void withdraw(double amount);

    public void transfer(Account target, double amount) {
        if (amount <= 0) throw new IllegalArgumentException("Amount must
be positive");
        if (target == null) throw new IllegalArgumentException("Target
account cannot be null");
        if (balance < amount) throw new
IllegalArgumentException("Insufficient funds");

        withdraw(amount);
        target.deposit(amount);
        addTransaction("Transferred $" + amount + " to " +
target.accountNumber);
    }

    public double checkBalance() {
        return balance;
```

```
        }

        protected void addTransaction(String info) {
            transactionHistory.add(java.time.LocalDateTime.now() + " - " +
    info);
        }

        public void showTransactionHistory() {
            transactionHistory.forEach(System.out::println);
        }
}
```

SavingAccount.java

```
    package banking;

    public class SavingsAccount extends Account implements BankOperations {
        private static final double MIN_BALANCE = 1000.0;

        public SavingsAccount(String accountNumber) {
            super(accountNumber);
            addTransaction("Savings account opened");
        }

        @Override
        public void deposit(double amount) {
            if (amount <= 0) throw new IllegalArgumentException("Deposit
    amount must be positive");
            balance += amount;
            addTransaction("Deposited $" + amount);
        }

        @Override
        public void withdraw(double amount) {
            if (amount <= 0) throw new IllegalArgumentException("Withdrawal
    amount must be positive");
            if (balance - amount < MIN_BALANCE) {
                throw new IllegalArgumentException("Minimum balance
    requirement not met");
            }
            balance -= amount;
            addTransaction("Withdrew $" + amount);
        }
}
```

Main.java

```
    import banking.SavingsAccount;

    public class Main {
```

```
    public static void main(String[] args) {
        SavingsAccount account = new SavingsAccount("SAV123456");

        account.deposit(1500);
        account.withdraw(200);
        account.showTransactionHistory();
    }
}
```

4.

CurrentAccount.java

```
package banking;

public class CurrentAccount extends Account implements BankOperations {
    private final double OVERDRAFT_LIMIT = 2000.0;

    public CurrentAccount(String accountNumber) {
        super(accountNumber);
        addTransaction("Current Account created with overdraft limit: $" +
OVERDRAFT_LIMIT);
    }

    @Override
    public void deposit(double amount) {
        if (amount <= 0) {
            addTransaction("Deposit failed: Amount must be positive");
            throw new IllegalArgumentException("Deposit amount must be
positive");
        }
        balance += amount;
        addTransaction(String.format("Deposited: $%.2f | New Balance:
$%.2f", amount, balance));
    }

    @Override
    public void withdraw(double amount) {
        if (amount <= 0) {
            addTransaction("Withdrawal failed: Amount must be positive");
            throw new IllegalArgumentException("Withdrawal amount must be
positive");
        }

        if (balance - amount < -OVERDRAFT_LIMIT) {
            addTransaction(String.format("Withdrawal failed: $%.2f exceeds
overdraft limit", amount));
            throw new IllegalArgumentException("Exceeds overdraft limit");
        }

        balance -= amount;
        addTransaction(String.format("Withdrew: $%.2f | New Balance: $%.2f",
amount, balance));
    }

    public double getAvailableBalance() {
        return balance + OVERDRAFT_LIMIT;
    }
}
```

5.

Customer.java

```java
package banking;

import java.util.ArrayList;
import java.util.List;

public class Customer {
    private final String customerId;
    private final String name;
    private final List<Account> accounts;

    public Customer(String customerId, String name) {
        if (customerId == null || customerId.trim().isEmpty()) {
            throw new IllegalArgumentException("Customer ID cannot be null
or empty");
        }
        if (name == null || name.trim().isEmpty()) {
            throw new IllegalArgumentException("Name cannot be null or
empty");
        }

        this.customerId = customerId;
        this.name = name.trim();
        this.accounts = new ArrayList<>();
    }

    public void addAccount(Account account) {
        if (account == null) {
            throw new IllegalArgumentException("Account cannot be null");
        }
        accounts.add(account);
    }

    public List<Account> getAccounts() {
        return new ArrayList<>(accounts); // Return a copy to maintain
encapsulation
    }

    public String getCustomerId() {
        return customerId;
    }

    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Customer{" +
               "customerId='" + customerId + '\'' +
               ", name='" + name + '\'' +
               ", accounts=" + accounts.size() +
               '}';
    }
}
```