Here's a complete **real-time banking application in Core Java** that incorporates **all major Java 8 features** without using Spring or any third-party frameworks. This design focuses solely on **Java 8 constructs** like:

* ✅ Lambda Expressions
* ✅ Stream API
* ✅ Optional
* ✅ Functional Interfaces
* ✅ Default & Static Methods in Interfaces
* ✅ Method References
* ✅ java.time API

**🏦 Real-Time Banking App — Core Java + Java 8 Features**

**📁 Project Structure**

css

CopyEdit

banking-app/

├── model/

│ ├── Customer.java

│ ├── Account.java

│ ├── Transaction.java

├── service/

│ ├── BankService.java

│ └── InterestCalculator.java

├── util/

│ ├── TransactionFilter.java

├── Main.java

**✅ Feature Highlights**

| **Functionality** | **Java 8 Feature In Use** |
| --- | --- |
| Account Filtering, Sorting | Stream API, Lambda |
| Null-safe lookups | Optional |
| Transaction validation | Functional Interface |
| Interest calculation | Default Method |
| Static utilities | Static Method |
| Logs & processing | Method References |
| Time-stamping transactions | java.time API |

**🔹 1. Customer.java**

public class Customer {

private String id;

private String name;

private Account account;

public Customer(String id, String name) {

this.id = id;

this.name = name;

this.account = new Account(id);

}

public Account getAccount() {

return account;

}

// Getters

}

**🔹 2. Account.java**

public class Account {

private String accountNumber;

private double balance;

private List<Transaction> transactions = new ArrayList<>();

public Account(String accountNumber) {

this.accountNumber = accountNumber;

}

public void deposit(double amount) {

balance += amount;

transactions.add(new Transaction("DEPOSIT", amount, LocalDateTime.now()));

}

public void withdraw(double amount) {

if (amount > balance) throw new IllegalArgumentException("Insufficient Balance");

balance -= amount;

transactions.add(new Transaction("WITHDRAW", amount, LocalDateTime.now()));

}

public Stream<Transaction> getTransactionStream() {

return transactions.stream();

}

public double getBalance() {

return balance;

}

}

**🔹 3. Transaction.java**

public class Transaction {

private String type;

private double amount;

private LocalDateTime date;

public Transaction(String type, double amount, LocalDateTime date) {

this.type = type;

this.amount = amount;

this.date = date;

}

@Override

public String toString() {

return type + " of ₹" + amount + " on " + date;

}

public double getAmount() { return amount; }

public String getType() { return type; }

public LocalDateTime getDate() { return date; }

}

**🔹 4. TransactionFilter.java — Functional Interface**

@FunctionalInterface

public interface TransactionFilter {

boolean test(Transaction t);

static TransactionFilter byMinAmount(double min) {

return tx -> tx.getAmount() >= min;

}

default TransactionFilter and(TransactionFilter other) {

return tx -> this.test(tx) && other.test(tx);

}

}

**🔹 5. InterestCalculator.java — Interface with Default Method**

public interface InterestCalculator {

default double calculateInterest(double balance, double rate) {

return balance \* rate / 100;

}

static double yearlyInterest(double balance, double rate) {

return balance \* Math.pow(1 + rate / 100, 1) - balance;

}

}

**🔹 6. BankService.java**

public class BankService implements InterestCalculator {

private Map<String, Customer> customers = new HashMap<>();

public void addCustomer(String id, String name) {

customers.putIfAbsent(id, new Customer(id, name));

}

public Optional<Customer> findCustomer(String id) {

return Optional.ofNullable(customers.get(id));

}

public void deposit(String id, double amount) {

findCustomer(id).ifPresent(c -> c.getAccount().deposit(amount));

}

public void withdraw(String id, double amount) {

findCustomer(id).ifPresent(c -> c.getAccount().withdraw(amount));

}

public List<Transaction> filterTransactions(String id, TransactionFilter filter) {

return findCustomer(id)

.map(c -> c.getAccount().getTransactionStream()

.filter(filter::test)

.collect(Collectors.toList()))

.orElse(Collections.emptyList());

}

public double checkInterest(String id, double rate) {

return findCustomer(id)

.map(c -> calculateInterest(c.getAccount().getBalance(), rate))

.orElse(0.0);

}

}

**🔹 7. Main.java — Execution**

public class Main {

public static void main(String[] args) {

BankService bank = new BankService();

bank.addCustomer("C001", "Preety Singh");

bank.deposit("C001", 5000);

bank.deposit("C001", 15000);

bank.withdraw("C001", 2000);

System.out.println("\n💰 All Transactions:");

bank.findCustomer("C001").ifPresent(c ->

c.getAccount().getTransactionStream()

.forEach(System.out::println)

);

System.out.println("\n📌 Filter: Transactions > ₹10,000:");

List<Transaction> filtered = bank.filterTransactions("C001", TransactionFilter.byMinAmount(10000));

filtered.forEach(System.out::println);

System.out.println("\n📈 Yearly Interest at 6%: ₹" +

bank.checkInterest("C001", 6.0)

);

}

}

**📘 Summary of Java 8 Usage**

| **Feature** | **Location Used** |
| --- | --- |
| Lambda Expressions | .filter(tx -> tx.getAmount() > 10000) |
| Method References | System.out::println |
| Stream API | Filtering/sorting transactions |
| Optional | Customer lookup |
| Functional Interface | TransactionFilter |
| Default & Static Methods | InterestCalculator interface |
| java.time API | LocalDateTime.now() for timestamp |