```
package com.mylendingapp;

import java.util.*;

class Borrower {
    private String borrowerId;
    private String name;
    private String region;
    private int creditScore;
    private List<Loan> activeLoans;

    public Borrower(String borrowerId, String name, String region, int creditScore) {
        this.borrowerId = borrowerId;
        this.name = name;
        this.region = region;
        this.creditScore = creditScore;
        this.activeLoans = new ArrayList<>();
    }

    // Getters and setters
    public String getBorrowerId() { return borrowerId; }
    public String getName() { return name; }
    public String getRegion() { return region; }
    public int getCreditScore() { return creditScore; }
    public List<Loan> getActiveLoans() { return activeLoans; }

    public void addLoan(Loan loan) {
        activeLoans.add(loan);
    }

    public void showSummary() {
        System.out.println("Borrower: " + name + ", Region: " + region + ", Credit Score: " + creditScore);
        if (activeLoans.isEmpty()) {
            System.out.println("No active loans.");
        } else {
            System.out.println("Active loans:");
            for (Loan loan : activeLoans) {
                System.out.println("  Loan ID: " + loan.getLoanId() + ", Principal: " + loan.getPrincipal() +
                        ", Outstanding: " + loan.getOutstandingAmount() + ", Status: " + loan.getStatus());
            }
        }
    }
}

class Lender {
    private String lenderId;
    private String name;
    private double walletBalance;
    private List<Loan> portfolio;

    public Lender(String lenderId, String name, double walletBalance) {
        this.lenderId = lenderId;
        this.name = name;
        this.walletBalance = walletBalance;
        this.portfolio = new ArrayList<>();
    }

    // Getters and setters
    public String getLenderId() { return lenderId; }
    public String getName() { return name; }
    public double getWalletBalance() { return walletBalance; }
    public List<Loan> getPortfolio() { return portfolio; }

    public boolean deductBalance(double amount) {
        if (amount <= walletBalance) {
            walletBalance -= amount;
            return true;
        }
        return false;
    }

    public void addBalance(double amount) {
        walletBalance += amount;
    }

    public void addToPortfolio(Loan loan) {
        portfolio.add(loan);
    }

    public double computeReturns() {
        double totalReturns = 0;
```

```
        for (Loan loan : portfolio) {
            totalReturns += loan.computeReturns();
        }
        return totalReturns;
    }

    public void showSummary() {
        System.out.println("Lender: " + name + ", Wallet Balance: " + walletBalance);
        System.out.println("Portfolio:");
        if (portfolio.isEmpty()) {
            System.out.println(" No loans funded yet.");
        } else {
            for (Loan loan : portfolio) {
                System.out.println("  Loan ID: " + loan.getLoanId() + ", Funded Amount: " +
loan.getFundedAmount() +
                                   ", Status: " + loan.getStatus());
            }
        }
        System.out.println("Estimated Returns: " + computeReturns());
    }
}

class Loan {
    protected String loanId;
    protected double principal;
    protected double interestRate; // annual %
    protected int tenure; // months
    protected double fundedAmount;
protected String status; // "Listed", "Funded", "Disbursed", "Closed"

protected double amountRepaid;


public Loan(String loanId, double principal, double interestRate, int tenure) {

    this.loanId = loanId;

    this.principal = principal;

    this.interestRate = interestRate;

    this.tenure = tenure;

    this.fundedAmount = 0;

    this.status = "Listed";

    this.amountRepaid = 0;

}


public String getLoanId() { return loanId; }

public double getPrincipal() { return principal; }

public double getInterestRate() { return interestRate; }

public int getTenure() { return tenure; }

public double getFundedAmount() { return fundedAmount; }

public String getStatus() { return status; }

public double getAmountRepaid() { return amountRepaid; }


// Fund loan by fixed amount (method overloading)

public boolean fundLoan(double amount) {

    if (status.equals("Listed") && (fundedAmount + amount) <= principal) {

        fundedAmount += amount;

        if (fundedAmount == principal) {

            status = "Funded";
        }
        return true;
    }
    return false;
}

// Fund loan by percentage of principal
public boolean fundLoan(int percent) {
    double amount = (principal * percent) / 100.0;
    return fundLoan(amount);
}

public void disburse() {
    if (status.equals("Funded")) {
        status = "Disbursed";
        System.out.println("Loan " + loanId + " disbursed.");
    } else {
        System.out.println("Loan not fully funded yet.");
    }
}

public void repay(double amount) {
    if (status.equals("Disbursed")) {
        amountRepaid += amount;
        if (amountRepaid >= principal + computeTotalInterest()) {
            status = "Closed";
            System.out.println("Loan " + loanId + " fully repaid.");
        } else {
            System.out.println("Repayment recorded.");
        }
    } else {
        System.out.println("Loan not disbursed yet.");
    }
}

// Default loan schedule (can be overridden)
public void loanSchedule() {
    System.out.println("Loan schedule for loan ID " + loanId + ":");
    double monthlyPrincipal = principal / tenure;
    double monthlyInterest = (principal * (interestRate / 100)) / 12;
    for (int month = 1; month <= tenure; month++) {
        System.out.printf("Month %d: Principal = %.2f, Interest = %.2f%n", month, monthlyPrincipal,
monthlyInterest);
    }
}

public double computeTotalInterest() {
    // Simple interest for demo
    return (principal * interestRate * tenure) / (12 * 100);
}

public double computeReturns() {
    // Return interest on funded amount proportionally
    return (fundedAmount / principal) * computeTotalInterest();
}
```

```
    public double getOutstandingAmount() {
        return (principal + computeTotalInterest()) - amountRepaid;
    }
}

class EducationLoan extends Loan {
    private String institution;

    public EducationLoan(String loanId, double principal, double interestRate, int tenure, String
institution) {
        super(loanId, principal, interestRate, tenure);
        this.institution = institution;
    }

    // Override loanSchedule for education loan specifics
    @Override
    public void loanSchedule() {
        System.out.println("Education Loan schedule for loan ID " + loanId + ":");
        double monthlyPrincipal = principal / tenure;
        // Education loans might have interest-only periods (for demo assume half tenure interest-only)
        int interestOnlyMonths = tenure / 2;
        for (int month = 1; month <= tenure; month++) {
            if (month <= interestOnlyMonths) {
                double monthlyInterest = (principal * (interestRate / 100)) / 12;
                System.out.printf("Month %d: Interest Only = %.2f%n", month, monthlyInterest);
            } else {
                double monthlyPrincipalPay = principal / (tenure - interestOnlyMonths);
                double monthlyInterest = (principal * (interestRate / 100)) / 12;
                System.out.printf("Month %d: Principal = %.2f, Interest = %.2f%n", month, 
monthlyPrincipalPay, monthlyInterest);
            }
        }
    }
}

class BusinessLoan extends Loan {
    private String businessType;

    public BusinessLoan(String loanId, double principal, double interestRate, int tenure, String 
businessType) {
        super(loanId, principal, interestRate, tenure);
        this.businessType = businessType;
    }

    // Override loanSchedule for business loan specifics
    @Override
    public void loanSchedule() {
        System.out.println("Business Loan schedule for loan ID " + loanId + ":");
        // Business loans might have equal principal + interest monthly
        double monthlyPrincipal = principal / tenure;
        for (int month = 1; month <= tenure; month++) {
            double monthlyInterest = ((principal - (monthlyPrincipal * (month - 1))) * (interestRate / 100)) 
/ 12;
            System.out.printf("Month %d: Principal = %.2f, Interest = %.2f%n", month, monthlyPrincipal, 
monthlyInterest);
        }
    }
}

class LendingService {
    private Map<String, Borrower> borrowers;
    private Map<String, Lender> lenders;
    private Map<String, Loan> loans;

    public LendingService() {
        borrowers = new HashMap<>();
        lenders = new HashMap<>();
        loans = new HashMap<>();
    }

    public void registerBorrower(Borrower borrower) {
        if (borrowers.containsKey(borrower.getBorrowerId())) {
            System.out.println("Borrower ID already exists.");
        } else {
            borrowers.put(borrower.getBorrowerId(), borrower);
            System.out.println("Borrower registered.");
        }
    }

    public void registerLender(Lender lender) {
        if (lenders.containsKey(lender.getLenderId())) {
            System.out.println("Lender ID already exists.");
        } else {
```

```
        lenders.put(lender.getLenderId(), lender);

        System.out.println("Lender registered.");

    }

}


public Borrower findBorrowerById(String id) {

    return borrowers.get(id);

}


public Lender findLenderById(String id) {

    return lenders.get(id);

}


public Loan findLoanById(String id) {

    return loans.get(id);

}


public void addLoan(Loan loan, Borrower borrower) {

    if (loans.containsKey(loan.getLoanId())) {

        System.out.println("Loan ID already exists.");

        return;

    }

    loans.put(loan.getLoanId(), loan);

    borrower.addLoan(loan);

    System.out.println("Loan added and assigned to borrower " + borrower.getName());

}
```

```java
public void listLoans() {
    if (loans.isEmpty()) {
        System.out.println("No loans available.");
        return;
    }
    System.out.println("Loans:");
    for (Loan loan : loans.values()) {
        System.out.printf("Loan ID: %s, Principal: %.2f, Funded: %.2f, Status: %s%n",
                loan.getLoanId(), loan.getPrincipal(), loan.getFundedAmount(), loan.getStatus());
    }
}

// fundLoan overloaded for amount or percent
public void fundLoan(Lender lender, Loan loan, double amount) {
    if (!loan.getStatus().equals("Listed") && !loan.getStatus().equals("Funded")) {
        System.out.println("Loan not available for funding.");
        return;
    }
    if (lender.getWalletBalance() < amount) {
        System.out.println("Lender does not have enough balance.");
        return;
    }
    boolean funded = loan.fundLoan(amount);
    if (funded) {
        lender.deductBalance(amount);
        lender.addToPortfolio(loan);
        System.out.println("Loan funded with amount " + amount);
    } else {
        System.out.println("Funding amount exceeds loan principal or loan fully funded.");
    }
}

public void fundLoan(Lender lender, Loan loan, int percent) {
    double amount = (loan.getPrincipal() * percent) / 100.0;
    fundLoan(lender, loan, amount);
}

public void disburseLoan(Loan loan) {
    loan.disburse();
}

public void recordRepayment(Loan loan, double amount) {
    loan.repay(amount);
}

public void showBorrowersSummary() {
    for (Borrower borrower : borrowers.values()) {
        borrower.showSummary();
        System.out.println();
    }
}

public void showLendersSummary() {
    for (Lender lender : lenders.values()) {
            lender.showSummary();

            System.out.println();

        }

    }

}


public class LendingAppMain {

    private static LendingService service = new LendingService();

    private static Scanner scanner = new Scanner(System.in);


    public static void main(String[] args) {

        initData();


        while (true) {

            printMenu();

            int choice = getIntInput("Enter your choice: ");

            switch (choice) {

                case 1: registerBorrower(); break;

                case 2: registerLender(); break;

                case 3: addLoan(); break;

                case 4: listLoans(); break;

                case 5: fundLoan(); break;

                case 6: disburseLoan(); break;

                case 7: recordRepayment(); break;

                case 8: showBorrowersSummary(); break;

                case 9: showLendersSummary(); break;

                case 0:
```

```
                System.out.println("Exiting application. Goodbye!");
                System.exit(0);
            default:
                System.out.println("Invalid choice.");
                break;
        }
    }
}

private static void printMenu() {
    System.out.println("\nMenu:");
    System.out.println("1. Register Borrower");
    System.out.println("2. Register Lender");
    System.out.println("3. Add Loan for Borrower");
    System.out.println("4. List Loans Available");
    System.out.println("5. Fund Loan");
    System.out.println("6. Disburse Loan");
    System.out.println("7. Record Repayment");
    System.out.println("8. Show Borrowers Summary");
    System.out.println("9. Show Lenders Summary");
    System.out.println("0. Exit");
}

private static void initData() {
    // Add some initial borrowers
    Borrower b1 = new Borrower("B001", "Alice", "East", 750);
    Borrower b2 = new Borrower("B002", "Bob", "West", 680);
    service.registerBorrower(b1);

    service.registerBorrower(b2);


    // Add some initial lenders

    Lender l1 = new Lender("L001", "Carol", 10000);

    Lender l2 = new Lender("L002", "Dave", 5000);

    service.registerLender(l1);

    service.registerLender(l2);


    // Add loans for Alice and Bob

    Loan edLoan = new EducationLoan("ED001", 5000, 5, 24, "ABC University");

    Loan busLoan = new BusinessLoan("BUS001", 10000, 8, 36, "Retail");

    service.addLoan(edLoan, b1);

    service.addLoan(busLoan, b2);

  }


  private static void registerBorrower() {

    System.out.println("Register New Borrower:");

    String id = getStringInput("Enter Borrower ID: ");

    String name = getStringInput("Enter Name: ");

    String region = getStringInput("Enter Region: ");

    int score = getIntInput("Enter Credit Score: ");

    Borrower borrower = new Borrower(id, name, region, score);

    service.registerBorrower(borrower);

  }


  private static void registerLender() {
    System.out.println("Register New Lender:");

    String id = getStringInput("Enter Lender ID: ");

    String name = getStringInput("Enter Name: ");

    double balance = getDoubleInput("Enter Wallet Balance: ");

    Lender lender = new Lender(id, name, balance);

    service.registerLender(lender);

}


private static void addLoan() {

    System.out.println("Add Loan:");

    String loanType = getStringInput("Enter Loan Type (Education/Business): ").toLowerCase();

    String loanId = getStringInput("Enter Loan ID: ");

    double principal = getDoubleInput("Enter Principal Amount: ");

    double interest = getDoubleInput("Enter Interest Rate (% per annum): ");

    int tenure = getIntInput("Enter Tenure (months): ");

    String borrowerId = getStringInput("Enter Borrower ID: ");

    Borrower borrower = service.findBorrowerById(borrowerId);

    if (borrower == null) {

        System.out.println("Borrower not found.");

        return;

    }


    Loan loan = null;

    if ("education".equals(loanType)) {

        String institution = getStringInput("Enter Institution Name: ");

        loan = new EducationLoan(loanId, principal, interest, tenure, institution);

    } else if ("business".equals(loanType)) {
        String bType = getStringInput("Enter Business Type: ");
        loan = new BusinessLoan(loanId, principal, interest, tenure, bType);
    } else {
        System.out.println("Invalid loan type.");
        return;
    }

    service.addLoan(loan, borrower);
}

private static void listLoans() {
    service.listLoans();
}

private static void fundLoan() {
    System.out.println("Fund Loan:");
    String lenderId = getStringInput("Enter Lender ID: ");
    String loanId = getStringInput("Enter Loan ID: ");
    Lender lender = service.findLenderById(lenderId);
    Loan loan = service.findLoanById(loanId);
    if (lender == null || loan == null) {
        System.out.println("Lender or Loan not found.");
        return;
    }

    System.out.println("Fund by:");
    System.out.println("1. Amount");
    System.out.println("2. Percentage");
    int option = getIntInput("Choose option: ");
    if (option == 1) {
        double amount = getDoubleInput("Enter amount to fund: ");
        service.fundLoan(lender, loan, amount);
    } else if (option == 2) {
        int percent = getIntInput("Enter percent to fund: ");
        service.fundLoan(lender, loan, percent);
    } else {
        System.out.println("Invalid option.");
    }
}

private static void disburseLoan() {
    System.out.println("Disburse Loan:");
    String loanId = getStringInput("Enter Loan ID: ");
    Loan loan = service.findLoanById(loanId);
    if (loan == null) {
        System.out.println("Loan not found.");
        return;
    }
    service.disburseLoan(loan);
}

private static void recordRepayment() {
    System.out.println("Record Repayment:");
    String loanId = getStringInput("Enter Loan ID: ");
```

```java
        Loan loan = service.findLoanById(loanId);
        if (loan == null) {
            System.out.println("Loan not found.");
            return;
        }
        double amount = getDoubleInput("Enter repayment amount: ");
        service.recordRepayment(loan, amount);
    }

    private static void showBorrowersSummary() {
        service.showBorrowersSummary();
    }

    private static void showLendersSummary() {
        service.showLendersSummary();
    }

    private static String getStringInput(String prompt) {
        System.out.print(prompt);
        return scanner.nextLine().trim();
    }

    private static int getIntInput(String prompt) {
        while (true) {
            try {
                System.out.print(prompt);
                int value = Integer.parseInt(scanner.nextLine().trim());
                return value;

            } catch (NumberFormatException e) {

                System.out.println("Please enter a valid integer.");

            }

        }

    }


    private static double getDoubleInput(String prompt) {

        while (true) {

            try {

                System.out.print(prompt);

                double value = Double.parseDouble(scanner.nextLine().trim());

                return value;

            } catch (NumberFormatException e) {

                System.out.println("Please enter a valid number.");

            }

        }

    }

}
```