

QUESTION:

Design and implement a console-based Crowdfunding app to create campaigns, accept pledges, track funding progress, and release funds using OOP in Java.

Requirements:

1. Create at least 4 classes:

- o User – userId, name, email, role (Creator/Backer), wallet.
- o Campaign – campaignId, title, goalAmount, deadline, status, pledges.
- o Pledge – pledgeId, backer, amount, rewardTier, date.
- o FundingService – creates campaigns, records pledges, computes progress, settles payouts.

2. Each class must include:

- o ≥ 4 instance/static variables.
- o A constructor to initialize values.
- o ≥ 5 methods (getters/setters, createCampaign(), addPledge(), progress(), settle()).

3. Demonstrate OOPS Concepts:

- o Inheritance → Creator extends User, VIPBacker extends User with perks.
- o Method Overloading → addPledge() by amount only or amount+rewardTier/coupon.
- o Method Overriding → custom settle() rules for flexible vs all-or-nothing campaigns.
- o Polymorphism → store users as User and apply role-specific behaviors.
- o Encapsulation → guard wallet balances and campaign states.

4. Write a Main class (CrowdfundAppMain) to test:

- o Register users, launch campaigns, take pledges.
- o Show progress %, reach goal, settle payouts/refunds.
- o Print top campaigns and backer contribution reports.

SOURCE CODE:

```
//UserRole

package basic;

    public enum UserRole { CREATOR, BACKER; }

    class User {
        protected String userId, name, email;
        protected UserRole role;
        protected double wallet;

        public User(String userId, String name, String email, UserRole role, double
wallet) {

            this.userId = userId;
            this.name = name;
            this.email = email;
            this.role = role;
            this.wallet = wallet;
        }

        public String getUserId() { return userId; }
        public String getName() { return name; }
        public String getEmail() { return email; }
        public UserRole getRole() { return role; }
        public double getWallet() { return wallet; }
        public void addToWallet(double amt) { wallet += amt; }
        public boolean deductFromWallet(double amt) {
            if(wallet >= amt) { wallet -= amt; return true; }
            return false;
        }
    }

    class Creator extends User {
        public Creator(String userId, String name, String email, double wallet) {
```

```

        super(userId, name, email, UserRole.CREATOR, wallet);
    }
}

class VIPBacker extends User {
    private double perks; // e.g. 10 = 10% cashback
    public VIPBacker(String userId, String name, String email, double wallet,
double perks) {
        super(userId, name, email, UserRole.BACKER, wallet);
        this.perks = perks;
    }
    public double getPerks() { return perks; }
}

```

//Pledge

```

package basic;
import java.time.LocalDate;
import java.util.*;
public class Pledge {
    private String pledgeId;
    private User backer;
    private double amount;
    private String rewardTier;
    private LocalDate date;
    public Pledge(String pledgeId, User backer, double amount, String rewardTier,
LocalDate date) {
        this.pledgeId = pledgeId; this.backer = backer; this.amount = amount;
this.rewardTier = rewardTier; this.date = date;
    }
    public String getPledgeId() { return pledgeId; }
    public User getBacker() { return backer; }
    public double getAmount() { return amount; }
}

```

```

        public String getRewardTier() { return rewardTier; }

        public LocalDate getDate() { return date; }
    }

    enum CampaignStatus { ACTIVE, SUCCESSFUL, FAILED, SETTLED; }

    class Campaign {

        protected String campaignId, title;

        protected double goalAmount, currentAmount;

        protected LocalDate deadline;

        protected CampaignStatus status;

        protected List<Pledge> pledges;

        protected Creator owner;

        public Campaign(String campaignId, String title, double goalAmount, LocalDate
deadline, Creator owner) {

            this.campaignId = campaignId;

            this.title = title;

            this.goalAmount = goalAmount;

            this.currentAmount = 0;

            this.deadline = deadline;

            this.status = CampaignStatus.ACTIVE;

            this.pledges = new ArrayList<>();

            this.owner = owner;

        }

        public String getCampaignId() { return campaignId; }

        public String getTitle() { return title; }

        public double getGoalAmount() { return goalAmount; }

        public double getCurrentAmount() { return currentAmount; }

        public Creator getOwner() { return owner; }

        public List<Pledge> getPledges() { return pledges; }

        public CampaignStatus getStatus() { return status; }

        public void setStatus(CampaignStatus status) { this.status = status; }

        public boolean addPledge(User backer, double amount) {

```

```

        return addPledge(backer, amount, "Standard", null);
    }

    public boolean addPledge(User backer, double amount, String rewardTier, String
coupon) {
        if(status != CampaignStatus.ACTIVE || LocalDate.now().isAfter(deadline))
return false;

        double actualAmount = amount;

        if (backer instanceof VIPBacker) {
            double perk = ((VIPBacker) backer).getPerks();
            actualAmount -= amount * perk / 100.0; // simple "cashback" handled as
discount
        }

        if (coupon != null && coupon.equals("DISC10")) { actualAmount *= 0.90; }

        if (!backer.deductFromWallet(actualAmount)) return false;

        String pledgeId = "P" + (pledges.size() + 1);

        Pledge p = new Pledge(pledgeId, backer, actualAmount, rewardTier,
LocalDate.now());

        pledges.add(p);

        currentAmount += actualAmount;

        if (currentAmount >= goalAmount) status = CampaignStatus.SUCCESSFUL;

        return true;
    }

    public double progress() { return (currentAmount / goalAmount) * 100.0; }

    public boolean settle() {
        if(status == CampaignStatus.SUCCESSFUL) {
            owner.addToWallet(currentAmount);

            status = CampaignStatus.SETTLED;

            return true;
        } else if(LocalDate.now().isAfter(deadline) && currentAmount < goalAmount)
{
            for(Pledge p : pledges) { p.getBacker().addToWallet(p.getAmount()); }

            status = CampaignStatus.FAILED;

```

```

        return false;
    }
    return false;
}
}

class FlexibleCampaign extends Campaign {
    public FlexibleCampaign(String campaignId, String title, double goalAmount,
        LocalDate deadline, Creator owner) {
        super(campaignId, title, goalAmount, deadline, owner);
    }
    @Override
    public boolean settle() {
        if (currentAmount > 0) {
            owner.addToWallet(currentAmount);
            status = CampaignStatus.SETTLED;
            return true;
        }
        return false;
    }
}

//FundingService
package basic;

import java.util.ArrayList;
import java.util.List;
import java.time.LocalDate;
import java.util.Map;
import java.util.HashMap;

public class FundingService {
    private List<User> users = new ArrayList<>();
    private List<Campaign> campaigns = new ArrayList<>();
    public void registerUser(User user) { users.add(user); }

```

```

        public Campaign createCampaign(String id, String title, double goal,
LocalDate deadline, Creator owner, boolean flexible) {

            Campaign c = flexible ? new FlexibleCampaign(id, title, goal,
deadline, owner)

                : new Campaign(id, title, goal, deadline, owner);

            campaigns.add(c); return c;

        }

        public boolean addPledge(Campaign c, User backer, double amount)
{ return c.addPledge(backer, amount); }

        public boolean addPledge(Campaign c, User backer, double amount,
String rewardTier, String coupon) {

            return c.addPledge(backer, amount, rewardTier, coupon);

        }

        public double progress(Campaign c) { return c.progress(); }

        public boolean settleCampaign(String id) {

            for(Campaign c:campaigns) if(c.getCampaignId().equals(id))

                return c.settle();

            return false;

        }

        public List<Campaign> topCampaigns() {

            campaigns.sort((a, b) -> Double.compare(b.progress(),
a.progress()));

            return campaigns.subList(0, Math.min(campaigns.size(), 3));

        }

        public Map<User, Double> backerReport() {

            Map<User, Double> res = new HashMap<>();

            for(Campaign c : campaigns)

                for(Pledge p : c.getPledges())

                    res.put(p.getBacker(), res.getOrDefault(p.getBacker(), 0.0) +

p.getAmount());

            return res;

        }

```

```
public User findUser(String id) { for(User u: users) if(u.getUserId().equals(id)) return u;
return null; }
```

```
        public Campaign findCampaign(String id) { for(Campaign c:
campaigns) if(c.getCampaignId().equals(id)) return c; return null; }
```

```
    }
```

```
// CrowdfundAppMain
```

```
package basic;
```

```
    import java.time.LocalDate;
```

```
    import java.util.*;
```

```
    public class CrowdfundAppMain {
```

```
        public static void main(String[] args) {
```

```
            FundingService service = new FundingService();
```

```
            Creator alice = new Creator("U1", "Alice", "alice@demo.com", 1000);
```

```
            User bob = new User("U2", "Bob", "bob@demo.com", UserRole.BACKER,
500);
```

```
            VIPBacker charlie = new VIPBacker("U3", "Charlie", "charlie@demo.com",
1000, 10); // 10% perks
```

```
            service.registerUser(alice);
```

```
            service.registerUser(bob);
```

```
            service.registerUser(charlie);
```

```
            Campaign lamp = service.createCampaign("C1", "Solar Lamp", 800,
LocalDate.now().plusDays(7), alice, false);
```

```
            Campaign bag = service.createCampaign("C2", "Eco Bag", 500,
LocalDate.now().plusDays(5), alice, true);
```

```
            service.addPledge(lamp, bob, 300); // basic
```

```
            service.addPledge(lamp, charlie, 600, "Deluxe", "DISC10"); // VIP and coupon
```

```
            service.addPledge(bag, charlie, 200, "Standard", null);
```

```
            System.out.printf("Lamp progress: %.2f%%\n", lamp.progress());
```

```
            System.out.printf("Bag progress: %.2f%%\n", bag.progress());
```

```
            System.out.println("Lamp settled: " + service.settleCampaign("C1"));
```

```
            System.out.println("Bag settled: " + service.settleCampaign("C2"));
```

```
            System.out.println("--- Top Campaigns ---");
```



```

    for (Campaign c : service.topCampaigns())

        System.out.printf("%s (%.2f%% funded)%n", c.getTitle(), c.progress());

    System.out.println("--- Backer Contribution Report ---");

    for (Map.Entry<User, Double> entry : service.backerReport().entrySet()) {

        System.out.printf("%s: %.2f%n", entry.getKey().getName(), entry.getValue());

    }

}

}

```

OUTPUT:

The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure with packages like `src`, `basic`, and `CampaignStatus.java`.
- Source Editor:** Displays the code for `CrowdfundAppMain.java`, showing the registration of users, creation of campaigns, and addition of pledges.
- Console:** Shows the output of the application, including campaign progress and backer contribution reports.

The output in the console is as follows:

```

<terminated> CrowdfundAppMain [Java Application] C:\Program Files\Java\jdk1.8.0_131\bin\javaw.exe (10-Sept-2025, 8:40:46 pm - 8:40:48 pm)
Lamp progress: 98.25%
Bag progress: 36.00%
Lamp settled: false
Bag settled: true
--- Top Campaigns ---
Solar Lamp (98.25% funded)
Eco Bag (36.00% funded)
--- Backer Contribution Report ---
Bob: 300.00
Charlie: 666.00

```

GITHUB LINK: