Design and implement a console-Based Referral & rewards system to generate referral codes, track signups, award points, and redeem rewards using OOP in java.

Requirements:

1. **Create at least 4 Classes:**

   * User - userId,name,email, referralCode, points

   * Referral -refId, inviter, inviteeEmail, status, bonus

   * Reward - rewardId, name, pointsRequired, inventory

   * ReferralService - code generation,tracking, awarding, redemption

2. **Each class must include:**

   * >= 4 instance/static variables

   * A constructor to initialize values.

   * > = methods (getters/setters, generateCode(), trackSignup(), awardPoints(), . . redeem())

3. **Demostrate OOPS concepts:**

   * Inheritance - vocherReaward/GiftReward extends Reward with rules.

   * Method Overloading - redeem() by rewardId or by name, with/without promo

   * Method overriding - different consumeInventory()/deliver() per reward type

   * Polymorphism - handle rewards as Reward adn dispatch delivery dynamically

   * Encapsulation - protect points and redemption state.

4. **Write Main class (ReferralAppMain)to test:**

   * Create users, generate codes, track signups, award points.

   * Redeem rewards and print referral leaderboards/monthly summaries.

```
import java.util.*;
// ====================== USER CLASS ======================
class User {
  private String userId;
  private String name;
  private String email;
  private String referralCode;
  private int points;

  public User(String userId, String name, String email) {
    this.userId = userId;
    this.name = name;
    this.email = email;
    this.referralCode = generateReferralCode();
    this.points = 0;
  }

  private String generateReferralCode() {
    return name.substring(0, 2).toUpperCase() + new Random().nextInt(9999);
  }

  public String getReferralCode() {
    return referralCode;
  }

  public String getName() {
    return name;
  }
```

```
    public String getUserId() {
        return userId;
    }

    public int getPoints() {
        return points;
    }

    public void addPoints(int pts) {
        this.points += pts;
    }

    public void deductPoints(int pts) {
        if (this.points >= pts) {
            this.points -= pts;
        }
    }

    @Override
    public String toString() {
        return "User{" + "userId='" + userId + '\'' +
                ", name='" + name + '\'' +
                ", email='" + email + '\'' +
                ", referralCode='" + referralCode + '\'' +
                ", points=" + points + '}';
    }
}
```

```
// ====================== REFERRAL CLASS ======================
class Referral {
    private String refId;
    private User inviter;
    private String inviteeEmail;
    private String status; // PENDING, SIGNEDUP
    private int bonus;
    public Referral(String refId, User inviter, String inviteeEmail) {
        this.refId = refId;
        this.inviter = inviter;
        this.inviteeEmail = inviteeEmail;
        this.status = "PENDING";
        this.bonus = 50;
    }

    public void markSignedUp() {
        this.status = "SIGNEDUP";
        inviter.addPoints(bonus);
    }

    public String getStatus() {
        return status;
    }

    public User getInviter() {
        return inviter;
    }
}
```

```
// ====================== REWARD CLASS =======================
abstract class Reward {
  protected String rewardId;
  protected String name;
  protected int pointsRequired;
  protected int inventory;

  public Reward(String rewardId, String name, int pointsRequired, int inventory) {
    this.rewardId = rewardId;
    this.name = name;
    this.pointsRequired = pointsRequired;
    this.inventory = inventory;
  }

  public String getRewardId() {
    return rewardId;
  }

  public String getName() {
    return name;
  }

  public int getPointsRequired() {
    return pointsRequired;
  }

  public boolean isAvailable() {
    return inventory > 0;
  }
   public abstract void deliver();
  public void consumeInventory() {
    if (inventory > 0) inventory--;
  }
}
// ====================== VOUCHER REWARD ======================
class VoucherReward extends Reward {
  public VoucherReward(String rewardId, String name, int pointsRequired, int inventory)
{
    super(rewardId, name, pointsRequired, inventory);
  }

  @Override
  public void deliver() {
    System.out.println("Voucher " + name + " delivered via Email.");
  }
}
// ====================== GIFT REWARD ======================
class GiftReward extends Reward {
  public GiftReward(String rewardId, String name, int pointsRequired, int inventory) {
    super(rewardId, name, pointsRequired, inventory);
  }

  @Override
  public void deliver() {
    System.out.println("Gift " + name + " shipped to user address.");
  }}
```

```
// ======================= REFERRAL SERVICE =======================
class ReferralService {
  private Map<String, User> users = new HashMap<>();
  private List<Referral> referrals = new ArrayList<>();
  private List<Reward> rewards = new ArrayList<>();


  // User management
  public User registerUser(String userId, String name, String email) {
    User u = new User(userId, name, email);
    users.put(userId, u);
    return u;
  }
  public void createReferral(User inviter, String inviteeEmail) {
    String refId = "REF" + (referrals.size() + 1);
    Referral r = new Referral(refId, inviter, inviteeEmail);
    referrals.add(r);
    System.out.println("Referral created by " + inviter.getName() + " for " + inviteeEmail);
  }


  public void trackSignup(String inviteeEmail) {
    for (Referral r : referrals) {
      if (r.getStatus().equals("PENDING") && r.inviteeEmail.equals(inviteeEmail)) {
        r.markSignedUp();
        System.out.println("Signup tracked! Points awarded to " +
r.getInviter().getName());
        break;
      }
    }
  }
  public void addReward(Reward reward) {
    rewards.add(reward);
  }

  public void redeem(User user, String rewardId) {
    for (Reward r : rewards) {
      if (r.getRewardId().equals(rewardId)) {
        processRedemption(user, r);
        return;
      }
    }
    System.out.println("Reward not found.");
  }

  public void redeem(User user, String rewardName, boolean withPromo) {
    for (Reward r : rewards) {
      if (r.getName().equalsIgnoreCase(rewardName)) {
        if (withPromo) {
          System.out.println("Promo applied! 10% discount on points.");
          r.pointsRequired *= 0.9;
        }
        processRedemption(user, r);
        return;
      }
    }
    System.out.println("Reward not found.");
  }
```

```
    private void processRedemption(User user, Reward r) {
        if (!r.isAvailable()) {
            System.out.println("Reward out of stock!");
            return;
        }
        if (user.getPoints() >= r.getPointsRequired()) {
            user.deductPoints(r.getPointsRequired());
            r.consumeInventory();
            r.deliver(); // Polymorphic dispatch
            System.out.println(user.getName() + " redeemed " + r.getName());
        } else {
            System.out.println("Not enough points to redeem " + r.getName());
        }
    }
    public void printLeaderboard() {
        System.out.println("\n===== Referral Leaderboard =====");
        users.values().stream()
                .sorted((u1, u2) -> Integer.compare(u2.getPoints(), u1.getPoints()))
                .forEach(u -> System.out.println(u.getName() + " - " + u.getPoints() + " points"));
    }
}
```

```
// ====================== MAIN APP ======================
public class ReferralAppMain {
  public static void main(String[] args) {
    ReferralService service = new ReferralService();


    User u1 = service.registerUser("U1", "Alice", "alice@mail.com");
    User u2 = service.registerUser("U2", "Bob", "bob@mail.com");


    service.createReferral(u1, "newuser@mail.com");
    service.trackSignup("newuser@mail.com");


    Reward rw1 = new VoucherReward("R1", "Amazon Voucher", 100, 5);
    Reward rw2 = new GiftReward("R2", "Coffee Mug", 150, 2);
    service.addReward(rw1);
    service.addReward(rw2);


    service.redeem(u1, "R1");
    service.redeem(u1, "Coffee Mug", true);


    service.printLeaderboard();
  }
}
```