Question 1: Refactor a Class Violating SRP

Refactor the `Report` class into separate classes, each with a single responsibility

```
class ReportGenerator:
    def __init__(self, data):
        self.data = data

    def generate_report(self):
        print("Report generated.")

class ReportSaver:
    def save_report(self, filename):
        print(f"Report saved to {filename}.")

# Testing
report = ReportGenerator("Some data")
report.generate_report()

saver = ReportSaver()
saver.save_report("report.txt")
```

```
Report generated.
Report saved to report.txt.
```

Question 2: Design a Class Hierarchy for Shapes

Refactor the `Shape` class into separate classes, ensuring that each class has a single responsibility. For example:

- A class for calculating area.
- A class for calculating perimeter

```
class Shape:
    def __init__(self, length, width):
        self.length = length
        self.width = width

class AreaCalculator:
    def calculate_area(self, shape):
        return shape.length * shape.width

class PerimeterCalculator:
    def calculate_perimeter(self, shape):
        return 2 * (shape.length + shape.width)

# Testing
rectangle = Shape(10, 5)

area_calc = AreaCalculator()
print(f"Area: {area_calc.calculate_area(rectangle)}")

perimeter_calc = PerimeterCalculator()
print(f"Perimeter: {perimeter_calc.calculate_perimeter(rectangle)}")
```

```
Area: 50
Perimeter: 30
```

Question 3: Refactor a User Management System

Refactor the `UserManager` class into separate classes, ensuring that each class has a single responsibility. For example:

- A class for user authentication.
- A class for user profile management.
- A class for email notifications.

```
class UserAuthenticator:
    def authenticate(self, username, password):
        print(f"User {username} authenticated.")

class UserProfileManager:
```

```
        def __init__(self, username):
            self.username = username

        def update_profile(self, new_username):
            self.username = new_username
            print(f"Profile updated. New username: {self.username}")

    class EmailService:
        def __init__(self, email):
            self.email = email

        def send_email(self, subject, message):
            print(f"Email sent to {self.email} with subject: {subject}")

    # Testing
    authenticator = UserAuthenticator()
    authenticator.authenticate("john_doe", "secure_password")

    profile_manager = UserProfileManager("john_doe")
    profile_manager.update_profile("john_updated")

    email_service = EmailService("john@example.com")
    email_service.send_email("Welcome!", "Hello, John!")
```

```
User john_doe authenticated.
Profile updated. New username: john_updated
Email sent to john@example.com with subject: Welcome!
```