Core Java-Assignment2

### 1. ****Product Price Comparison****

class Product {

String name;

double price;

Product(String name, double price) {

this.name = name;

this.price = price;

}

void comparePrice(Product otherProduct) {

if (this.price > otherProduct.price) {

System.out.println(this.name + " is more expensive.");

} else if (this.price < otherProduct.price) {

System.out.println(otherProduct.name + " is more expensive.");

} else {

System.out.println("Both products have the same price.");

}

}

public static void main(String[] args) {

Product product1 = new Product("Laptop", 1500.0);

Product product2 = new Product("Smartphone", 800.0);

product1.comparePrice(product2);

}

}

### 2. ****Vehicle Inheritance System****

class Vehicle {

void displayType() {

System.out.println("This is a vehicle.");

}

}

class Car extends Vehicle {

@Override

void displayType() {

System.out.println("This is a car.");

}

}

class Bike extends Vehicle {

@Override

void displayType() {

System.out.println("This is a bike.");

}

}

public class Main {

public static void main(String[] args) {

Vehicle vehicle = new Vehicle();

vehicle.displayType();

Vehicle car = new Car();

car.displayType();

Vehicle bike = new Bike();

bike.displayType();

}

}

### 3. ****Payment System using Method Overloading****

class PaymentSystem {

void processPayment(double amount) {

System.out.println("Payment of $" + amount + " processed via credit card.");

}

void processPayment(double amount, String cardNumber) {

System.out.println("Payment of $" + amount + " processed via credit card number " + cardNumber);

}

void processPayment(double amount, String cardNumber, String expiryDate) {

System.out.println("Payment of $" + amount + " processed via credit card " + cardNumber + " expiring on " + expiryDate);

}

public static void main(String[] args) {

PaymentSystem ps = new PaymentSystem();

ps.processPayment(100.0);

ps.processPayment(200.0, "1234-5678-9876-5432");

ps.processPayment(300.0, "1234-5678-9876-5432", "12/25");

}

}

### 4. ****Employee Salary Calculation using Method Overriding****

class Employee {

double calculateSalary() {

return 50000;

}

}

class Manager extends Employee {

@Override

double calculateSalary() {

return 70000;

}

}

public class Main {

public static void main(String[] args) {

Employee emp = new Employee();

System.out.println("Employee Salary: $" + emp.calculateSalary());

Manager mgr = new Manager();

System.out.println("Manager Salary: $" + mgr.calculateSalary());

}

}

### 5. ****Constructor Chaining in a Banking System****

class BankAccount {

String accountHolder;

double balance;

BankAccount(String accountHolder) {

this(accountHolder, 0.0); // Constructor chaining

}

BankAccount(String accountHolder, double balance) {

this.accountHolder = accountHolder;

this.balance = balance;

}

void displayDetails() {

System.out.println("Account Holder: " + accountHolder + ", Balance: $" + balance);

}

public static void main(String[] args) {

BankAccount account1 = new BankAccount("John Doe");

BankAccount account2 = new BankAccount("Jane Smith", 2000.0);

account1.displayDetails();

account2.displayDetails();

}

}

### 6. ****Secure User Data with Encapsulation****

class User {

private String username;

private String password;

public String getUsername() {

return username;

}

public void setUsername(String username) {

this.username = username;

}

public String getPassword() {

return password;

}

public void setPassword(String password) {

this.password = password;

}

public void displayUserInfo() {

System.out.println("Username: " + username);

}

public static void main(String[] args) {

User user = new User();

user.setUsername("john\_doe");

user.setPassword("securePassword123");

user.displayUserInfo();

}

}

### 7. ****Organizing a Library Management System****

class Book {

String title;

String author;

Book(String title, String author) {

this.title = title;

this.author = author;

}

void displayBookDetails() {

System.out.println("Title: " + title + ", Author: " + author);

}

}

class Library {

Book[] books;

Library(int capacity) {

books = new Book[capacity];

}

void addBook(Book book, int index) {

books[index] = book;

}

void displayBooks() {

for (Book book : books) {

if (book != null) {

book.displayBookDetails();

}

}

}

public static void main(String[] args) {

Library library = new Library(2);

library.addBook(new Book("The Java Programming Language", "James Gosling"), 0);

library.addBook(new Book("Clean Code", "Robert C. Martin"), 1);

library.displayBooks();

}

}

### 8. ****Online Shopping System using Abstraction****

abstract class Product {

abstract void displayDetails();

}

class Electronics extends Product {

void displayDetails() {

System.out.println("This is an electronic product.");

}

}

class Clothing extends Product {

void displayDetails() {

System.out.println("This is a clothing product.");

}

}

public class Main {

public static void main(String[] args) {

Product product1 = new Electronics();

Product product2 = new Clothing();

product1.displayDetails();

product2.displayDetails();

}

}

### 9. ****Using Inner Classes for Authentication****

class User {

private String username;

private String password;

User(String username, String password) {

this.username = username;

this.password = password;

}

class Authentication {

boolean authenticate(String inputUsername, String inputPassword) {

return username.equals(inputUsername) && password.equals(inputPassword);

}

}

public static void main(String[] args) {

User user = new User("john\_doe", "securePassword123");

User.Authentication auth = user.new Authentication();

if (auth.authenticate("john\_doe", "securePassword123")) {

System.out.println("Authentication successful.");

} else {

System.out.println("Authentication failed.");

}

}

}

### 10. ****Implementing an Anonymous Class for Sorting****

import java.util.Arrays;

public class AnonymousClassSort {

public static void main(String[] args) {

Integer[] numbers = {5, 2, 8, 1, 3};

Arrays.sort(numbers, new java.util.Comparator<Integer>() {

public int compare(Integer a, Integer b) {

return b - a; // Sorting in descending order

}

});

for (int num : numbers) {

System.out.print(num + " ");

}

}

}