SmartCliff
Career Mobility Solutions

Practice pool

Default Constructor:

1. Create a class named "Student" with a default constructor that sets the student's name as "Jane Smith" and grade as 10. Create an instance of the Student class and display the student's name and grade.
2. Create a class named "Customer" with a default constructor that initializes the customer's name as "Guest" and membership level as "Regular". Create an instance of the Customer class and print out the customer's name and membership level.

Parameterized Constructor:

1. Create a class named "Car" with a parameterized constructor that takes in the car's make and year as parameters. The constructor should initialize the car object with the provided values. Create an instance of the Car class with the make "Toyota" and year 2020 and print out the car's make and year.
2. Create a class named "Book" with a parameterized constructor that takes in the book's title and author as parameters. The constructor should initialize the book object with the provided values. Create an instance of the Book class with the title "The Great Gatsby" and author "F. Scott Fitzgerald" and display the book's title and author.

Constructor Overloading:

1. Create a class named "Date" with constructor overloading to support different ways of initializing a date object. Implement constructors that accept the day, month, and year as parameters, as well as constructors that accept only the day and month (assuming the current year) or only the day (assuming the current month and year). Create instances of the Date class using different constructor combinations and print out the dates.
2. Create a class named "Rectangle" with constructor overloading to support different ways of initializing a rectangle object. Implement constructors that accept the width and height as parameters, as well as constructors that accept only the width or only the height (assuming both are equal). Create instances

of the Rectangle class using different constructor combinations and display the rectangle's dimensions.

Encapsulation:

1. Create a Student class with a private field for the student's grade. Implement a read and write property for the grade to encapsulate access. Additionally, create a method to calculate the final grade based on assignments and exams. Demonstrate the usage of the Student class by creating instances, setting property values, calculating the final grade, and displaying the student's information.
2. Create a Book class with a private field for the availability status. Implement read and write properties for the availability to encapsulate access. Additionally, create methods to borrow and return the book. Demonstrate the usage of the Book class by creating instances, setting property values, borrowing and returning books, and displaying the availability status.

Inheritance:

Single Inheritance:

1. Create a base class called "Employee" with properties for the employee's name and salary. Implement a derived class called "Manager" that inherits from the Employee class and adds a property for the manager's bonus. Demonstrate the usage of the Manager class by creating instances, setting property values, and calculating the manager's total pay.
2. Create a base class called "BankAccount" with properties for the account holder's name and balance. Implement a derived class called "SavingsAccount" that inherits from the BankAccount class and adds a property for the savings account's interest rate. Demonstrate the usage of the SavingsAccount class by creating instances, setting property values, and performing transactions.

SmartCliff
Career Mobility Solutions

Practice pool

Multilevel Inheritance:

1. Create a base class called "Employee" with properties for the employee's name and salary. Implement a derived class called "Manager" that inherits from the Employee class and adds a property for the manager's department. Finally, implement a derived class called "Executive" that inherits from the Manager class and adds a property for the executive's bonus. Demonstrate the usage of the Executive class by creating instances, setting property values, and calculating the executive's total pay.
2. Create a base class called "Animal" with properties for the animal's name and age. Implement a derived class called "Mammal" that inherits from the Animal class and adds a property for the mammal's habitat. Finally, implement a derived class called "Dog" that inherits from the Mammal class and adds a property for the dog's breed. Demonstrate the usage of the Dog class by creating instances, setting property values, and displaying the dog's information.

Hierarchical Inheritance:

1. Create a base class called "Account" with common properties and methods such as "AccountNumber," "Balance," and "Withdraw." Implement derived classes such as "SavingsAccount" and "CheckingAccount" that inherit from the Account class. Add specific properties and methods to each derived class, such as "InterestRate" for savings accounts and "OverdraftLimit" for checking accounts. Demonstrate the usage of the Account class and its derived classes by creating instances, setting properties, invoking methods, and performing account operations.
2. Create a base class called "Product" with common properties and methods such as "Name," "Price," and "Description." Implement derived classes such as "Electronics" and "Clothing" that inherit from the Product class. Add specific properties and methods to each derived class, such as "WarrantyPeriod" for electronics and "Size" for clothing. Further, create subclasses such as "Television" and "T-Shirt" that inherit from the respective derived classes.

Demonstrate the usage of the Product class and its derived classes by creating instances, setting properties, invoking methods, and displaying the product details.

Hybrid Inheritance:

1. Create a base class called "Product" with common properties and methods such as "Name," "Price," and "Description." Implement derived classes such as "ElectronicProduct" and "ClothingProduct" that inherit from the Product class. Add specific properties and methods to each derived class, such as "WarrantyPeriod" for electronic products and "Size" for clothing products. Further, create a class called "DiscountedClothing" that inherits from both ClothingProduct and Product, allowing for discounted clothing products. Demonstrate the usage of the classes by creating instances, setting properties, invoking methods, and displaying product information.
2. Create a base class called "Person" with common properties and methods such as "Name," "Age," and "DisplayInformation." Implement derived classes such as "Student" and "Faculty" that inherit from the Person class. Add specific properties and methods to each derived class, such as "StudentId" for students and "Department" for faculty. Further, create a class called "TeachingAssistant" that inherits from both Student and Faculty, representing students who also work as teaching assistants. Demonstrate the usage of the classes by creating instances, setting properties, invoking methods, and displaying information about students, faculty, and teaching assistants.

Multiple Inheritance:

1. Create a base class called "Package" with common properties and methods such as "TrackingNumber," "Weight," and "CalculateShippingCost." Implement interfaces such as "IStandardPackage" and "IFragilePackage" that define specific properties and methods for standard packages and fragile packages, respectively. Implement classes such as "StandardPackage" and "FragilePackage" that inherit from the Package class and implement the respective interfaces. Demonstrate the usage of the classes by creating instances, setting properties, invoking methods, and simulating package deliveries.

2. Create a base class called "Recipe" with common properties and methods such as "Name," "CookingTime," and "Instructions." Implement interfaces such as "IDessertRecipe" and "IMainCourseRecipe" that define specific properties and methods for dessert recipes and main course recipes, respectively. Implement classes such as "DessertRecipe" and "MainCourseRecipe" that inherit from the Recipe class and implement the respective interfaces. Demonstrate the usage of the classes by creating instances, setting properties, invoking methods, and displaying recipe details.

Polymorphism:

Static Polymorphism:

1. Create a class called "GeometryUtils" with multiple static methods for performing calculations related to geometric shapes such as "Area" and "Perimeter" with different parameter combinations. Implement method overloading to handle different types of shapes and numbers of parameters. Demonstrate the usage of the GeometryUtils class by calling the overloaded methods with different arguments and displaying the calculated values.
2. Create a class called "DateUtils" with multiple static methods for date manipulation such as "AddDays" and "CalculateAge" with different parameter combinations. Implement method overloading to handle different date formats and calculations. Demonstrate the usage of the DateUtils class by calling the overloaded methods with different arguments and displaying the manipulated dates or calculated ages.

Dynamic Polymorphism:

1. Create a base class called "Shape" with a virtual method called "CompareTo" that compares the current shape with another shape and returns an integer value indicating their relative sizes. Implement derived classes such as "Rectangle" and "Circle" that override the "CompareTo" method to compare their specific dimensions. Demonstrate the usage of the Shape class and its

derived classes by creating instances, invoking the "CompareTo" method with different shapes, and displaying the comparison results.

2. Create a base class called "Message" with a virtual method called "Send" that simulates sending a message. Implement derived classes such as "Email" and "SMS" that override the "Send" method to provide specific sending mechanisms for each type of message. Demonstrate the usage of the Message class and its derived classes by creating instances, invoking the "Send" method, and simulating the sending of different types of messages.

Abstraction:

1. Create an abstract class called "Payment" with an abstract method called "ProcessPayment." Implement two derived classes called "CreditCardPayment" and "PayPalPayment" that inherit from the Payment class. Each derived class should provide its own implementation of the "ProcessPayment" method. Demonstrate the usage of the CreditCardPayment and PayPalPayment classes by creating instances and invoking the "ProcessPayment" method.
2. Create an abstract class called "FileStorage" with an abstract method called "SaveFile." Implement two derived classes called "LocalFileStorage" and "CloudFileStorage" that inherit from the FileStorage class. Each derived class should provide its own implementation of the "SaveFile" method. Demonstrate the usage of the LocalFileStorage and CloudFileStorage classes by creating instances and invoking the "SaveFile" method.

Interface:

1. Create an interface called "ISortAlgorithm" with a method signature for sorting an array. Implement classes such as "BubbleSort" and "QuickSort" that implement the ISortAlgorithm interface. Demonstrate the usage of the interface by creating instances of the classes and sorting different arrays using different sorting algorithms.

Practice pool

2. Create an interface called "IPaymentGateway" with method signatures for processing payments and refunding payments. Implement classes such as "PayPalGateway" and "StripeGateway" that implement the IPaymentGateway interface. Demonstrate the usage of the interface by creating instances of the classes, processing payments, and refunding payments using different payment gateways.