Question 1: Basic Types and Interfaces

Create an interface called `User` with properties for `id` (number), `name` (string), `email` (string), and `isActive` (boolean). Then, create a function called `createUser` that takes a user object of type `User` and returns it. Finally, write code to create a new user and call the function.

Question 2: Union Types and Type Guards

Create a type called `Input` that can be either a number or a string. Then write a function called `processInput` that takes an argument of type `Input` and returns a string. If the input is a number, convert it to a string and prepend "Number: " to it. If the input is already a string, prepend "String: " to it. Use type guards to check the type of input.

Question 3: Classes and Inheritance

Create a base class called `Vehicle` with properties for `make` (string), `model` (string), and `year` (number). Include a method called `getInfo()` that returns a string with the vehicle information. Then create two subclasses: `Car` and `Motorcycle`. The `Car` class should have an additional property for `doors` (number), and the `Motorcycle` class should have a property for `hasSidecar` (boolean). Override the `getInfo()` method in each subclass to include the additional information.

Question 4: Access Modifiers and Getters/Setters

Create a class called `BankAccount` with:

- A private property for `balance` (number)
- A private readonly property for `accountNumber` (string)
- A constructor that initializes both properties
- A getter method for balance
- A getter method for accountNumber
- A method called `deposit(amount: number)` that adds to the balance
- A method called `withdraw(amount: number)` that subtracts from the balance but prevents overdrafts by throwing an error if the amount is greater than the balance

Test the class by creating an account, making deposits and withdrawals, and trying to access the private properties directly.

Question 5: Abstract Classes

Create an abstract class called `Shape` with:

- A protected property for `color` (string)
- A constructor that sets the color
- An abstract method called `calculateArea()` that returns a number
- A concrete method called `getColor()` that returns the color

Then create two concrete classes that extend `Shape`:

- `Circle` with a property for `radius` (number)
- `Rectangle` with properties for `width` (number) and `height` (number)

Implement the `calculateArea()` method in each subclass. Then create instances of both shapes, calculate their areas, and get their colors.