
# JavaScript

STUDIA PODYPLOMOWE
POLITECHNIKA BIAŁOSTOCKA


HOMEWORK

# Curry

React example

# Factory

- Using factory design pattern, create different types of payment methods
- The payment factory should have the following types of payment methods: Credit Card, PayPal and Bank Transfer

# Factory

- Define a PaymentFactory class that will act as the factory creating payment methods

# Factory

- Implement the **createPayment** method on the PaymentFactory class.
- This method should accept two parameters:
  - type - that indicates the type of payment method to be created
  - paymentDetails - some object
- Should return instance of payment method based on the type

# Factory

- Define three payment classes: CreditCard, PayPal and BankTransfer
- Each payment class should have a constructor that accepts the necessary payment details (eg. card number, bank account, details)

# Proxy

- Using Proxy design pattern extend an object representing of a rectangle to calculate its area and perimeter

# Proxy

- Create a rectangle object that has 2 properties - width and height

# Proxy

- Create a proxy for the rectangle object
- The proxy should calculate the area and the perimeter whenever we access .area or .perimeter
- This should reflect any changes to width or height

# Proxy

- Disable changing .area or .perimeter

# Proxy

- Using Proxy design pattern extend an object representing of a user to store user password as “*”

# Proxy

- Create a user object with 3 properties: name, password, _password

# Proxy

- Create a proxy object that extends the user object and stores the password as “*”.
- When password is set, it should store the password in the _password property
- Also, the password should be stored as “*” in the password property

# Proxy

- If private _password is accessed it should throw an error

# Observer

- Implement the Observer design pattern to create a simple weather app that displays the current temperature

# Observer

- Your application should have a WeatherData class that represents the weather data and a WeatherDisplay class that displays the temperature

# Observer

- The WeatherData class should have a method called setTemperature that sets the current temperature and notifies all the observers about the change. The WeatherData class should also have a method called registerObserver that adds an observer to the list of observers

# Observer

- The WeatherDisplay class should have a method called update that updates the temperature displayed on the screen whenever the temperature changes

# Mediator

- Using Mediator Pattern create an auction house that allows multiple bidders to bid on a single item
- The highest bidder wins the auction

# Mediator

- There should be a Mediator class that manages the bidding process
- The Mediator object should keep track of the current highest bidder and their bid

# Mediator

- There should be a Bidder class that should be able to register with the Mediator object and receive updates on the current highest bid
- Bidders should be able to place bids through the Mediator object

# Adapter

- You have been tasked with implementing a date library in your project
- The library will be used to calculate the difference between two dates in hours

# Adapter

- Create index.js that will be using the functionality to calculate how many hours are between 24.01.2022 and now

# Adapter

- Create date.service.js that will have a function to calculate the difference in hours between 2 dates
- Use date-fns package: https://date-fns.org/v3.6.0/docs/differenceInHours
- npm init
- npm install date-fns

# Adapter

- Now switch the package to dayjs

  https://day.js.org/docs/en/installation/installation

- npm install dayjs

# Adapter

- You are using a Calculator class in your application which has an operation method. The method decides the result based on the provide parameters
- Start by implementing class Calculator with one method called operations
- Method should accept three parameters: x, y, operation

# Adapter

- Now imagine that your team decided to rewrite the calculator class from scratch.
- NewCalculator class, has been created, which no longer contains a single operation method, but all four basic operations function as separate methods.
- Now implement this class

# Adapter

- Create an Adapter class that will fit the new interface to the old one provided by the previous class

# Facade

- Based on Facade pattern create function that will return user with all his posts
- Use this api https://jsonplaceholder.typicode.com/

# Facade

- Create function to fetch user
- Create function to fetch all user posts

# Facade

- Create a faced function that will return user data and his posts

# HOMEWORK

- Memory game