1. **The class Movie is stated below. An instance of class Movie represents a film. This class has the following three properties: title, which is a String representing the title of the movie studio, which is a String representing the studio that made the movie rating, which is a String representing the rating of the movie (i.e. PG­13, R, etc)**

**a) Write a constructor for the class Movie, which takes a String representing the title of the movie, a String representing the studio, and a String representing the rating as its arguments, and sets the respective class properties to these values**.

constructor for the Movie class that takes three String arguments (title, studio, and rating) and sets the respective class properties to these values:

class Movie {

constructor(title, studio, rating) {

this.title = title;

this.studio = studio;

this.rating = rating;

}

}

// Example usage:

const movie1 = new Movie("The Shawshank Redemption", "Castle Rock Entertainment", "R");

// Accessing the properties of movie1

console.log("Title:", movie1.title);

console.log("Studio:", movie1.studio);

console.log("Rating:", movie1.rating);

You can use this constructor to create instances of the **Movie** class and initialize them with the specified title, studio, and rating values, just like in the example usage provided.

**b) The constructor for the class Movie will set the class property rating to "PG" as default when no rating is provided.**

class Movie {

constructor(title, studio, rating = "PG") {

this.title = title;

this.studio = studio;

this.rating = rating;

}

}

// Example usage with default rating "PG":

const movie1 = new Movie("The Shawshank Redemption", "Castle Rock Entertainment");

console.log("Title:", movie1.title);

console.log("Studio:", movie1.studio);

console.log("Rating:", movie1.rating); // Output: Rating: PG

// Example usage with a specified rating:

const movie2 = new Movie("Inception", "Warner Bros.", "PG-13");

console.log("Title:", movie2.title);

console.log("Studio:", movie2.studio);

console.log("Rating:", movie2.rating); // Output: Rating: PG-13

In this modified constructor, the rating parameter has a " PG " default value. If no rating is provided when creating a Movie object, it will default to "PG". However, if needed, you can still provide a specific rating, as shown in the example usage.

**c) Write a method getPG, which takes an array of base type Movie as its argument, and returns a new array of only those movies in the input array with a rating of "PG". You may assume the input array is full of Movie instances. The returned array need not be full.**

getPG method in the Movie class in JavaScript to filter an array of Movie instances and return a new array containing only those movies with a rating of "PG." Here's how you can do it:

class Movie {

constructor(title, studio, rating) {

this.title = title;

this.studio = studio;

this.rating = rating;

}

static getPG(movieArray) {

return movieArray.filter(movie => movie.rating === "PG");

}

}

// Example usage:

const movies = [

new Movie("Movie 1", "Studio A", "PG"),

new Movie("Movie 2", "Studio B", "PG-13"),

new Movie("Movie 3", "Studio C", "PG"),

new Movie("Movie 4", "Studio D", "R"),

];

const pgMovies = Movie.getPG(movies);

console.log(pgMovies);\

In this code, the getPG method is defined as a static method of the Movie class. It takes an array of Movie instances as its argument and uses the filter method to create a new array containing only those movies with a rating of "PG." Finally, it returns the filtered array.

When you call Movie.getPG(movies), it will return an array containing only the movies with a "PG" rating from the movies array.

**d) Write a piece of code that creates an instance of the class Movie with the title “Casino Royale”, the studio “Eon Productions”, and the rating “PG­13”**

To create an instance of the Movie class with the specified title, studio, and rating, you can write the following code in JavaScript:

class Movie {

constructor(title, studio, rating) {

this.title = title;

this.studio = studio;

this.rating = rating;

}

}

// Creating an instance of Movie with the specified details

const casinoRoyale = new Movie("Casino Royale", "Eon Productions", "PG-13");

console.log("Title:", casinoRoyale.title);

console.log("Studio:", casinoRoyale.studio);

console.log("Rating:", casinoRoyale.rating);

This code defines the Movie class and then creates an instance named casinoRoyale with the title "Casino Royale," the studio "Eon Productions," and the rating "PG-13." It then logs the properties of the casinoRoyale instance to the console.

**3. Write a “person” class to hold all the details**

class Person {

constructor(name, age, address, contactInfo) {

this.name = name;

this.age = age;

this.address = address;

this.contactInfo = contactInfo;

}

// Method to get the person's details as an object

getDetails() {

return {

Name: this.name,

Age: this.age,

Address: this.address,

ContactInfo: this.contactInfo,

};

}

// Method to update the person's address

updateAddress(newAddress) {

this.address = newAddress;

}

// Method to update contact information

updateContactInfo(newContactInfo) {

this.contactInfo = newContactInfo;

}

}

// Usage example:

const contactInfo = {

email: "john.doe@example.com",

phone: "123-456-7890",

};

const person1 = new Person("John Doe", 30, "123 Main St, City", contactInfo);

console.log(person1.getDetails());

// Output:

// {

// Name: 'John Doe',

// Age: 30,

// Address: '123 Main St, City',

// ContactInfo: { email: 'john.doe@example.com', phone: '123-456-7890' }

// }

// Update the address and contact information

person1.updateAddress("456 Elm St, Town");

person1.updateContactInfo({ email: "john.newemail@example.com", phone: "987-654-3210" });

console.log(person1.getDetails());

// Output after updates:

// {

// Name: 'John Doe',

// Age: 30,

// Address: '456 Elm St, Town',

// ContactInfo: { email: 'john.newemail@example.com', phone: '987-654-3210' }

// }

In this example, the "Person" class includes properties for the person's name, age, address, and contact information. It provides methods to get the person's details as an object, update the address, and update the contact information. You can create instances of this class and manipulate their details as demonstrated in the usage example.

**4. write a class to calculate the uber price**

Calculating the Uber price involves several factors such as distance, duration, base fare, and additional charges. Uber's pricing structure may vary by location and service type, so I'll provide you with a simplified JavaScript class that you can use as a starting point. You can adjust the pricing logic to match your specific requirements or the pricing structure of your region.

Here's a basic "UberPriceCalculator" class:

class UberPriceCalculator {

constructor(baseFare, costPerMile, costPerMinute, bookingFee) {

this.baseFare = baseFare;

this.costPerMile = costPerMile;

this.costPerMinute = costPerMinute;

this.bookingFee = bookingFee;

}

calculatePrice(distanceMiles, durationMinutes) {

const distanceCost = this.costPerMile \* distanceMiles;

const durationCost = this.costPerMinute \* durationMinutes;

const totalPrice = this.baseFare + distanceCost + durationCost + this.bookingFee;

return totalPrice;

}

}

// Usage example:

const uberX = new UberPriceCalculator(2.0, 1.5, 0.3, 1.0);

const distanceMiles = 5;

const durationMinutes = 15;

const totalPrice = uberX.calculatePrice(distanceMiles, durationMinutes);

console.log(`UberX Price: $${totalPrice.toFixed(2)}`);

In this example:

The UberPriceCalculator class is initialized with the base fare, cost per mile, cost per minute, and booking fee for a specific Uber service type (in this case, "UberX"). These values can vary by location and service type.

The calculatePrice method takes the distance in miles and the duration in minutes as input and calculates the total price based on the provided pricing parameters.

In the usage example, we create an instance of the UberPriceCalculator class for UberX, calculate the price for a 5-mile trip that takes 15 minutes, and then display the total price.

Please note that this is a simplified example, and real-world Uber pricing can be more complex due to factors like surge pricing, minimum fares, and other additional fees. You may need to adapt this class to match the specific pricing structure in your area or the type of Uber service you are interested in.