2021

Oppgave 1

a. *Lag kode for klassen Kjoretoy. Klassen skal ha to variabler makshastighet og kjorelengde. Disse forteller hvor raskt kjøretøyet kan kjøre og hvor langt kjøretøyet har*
kjørt.

```
class Kjoretoy{
    constructor(kjorelengde, makshastighet){
        this.makshastighet = makshastighet;
        this.kjorelengde = kjorelengde;
    }
}
```

b. *Lag kode som oppretter et kjøretøy som en instans av klassen Kjoretoy. Velg selv navn på*
dette objektet og passende verdier på alle variablene. Skriv deretter ut verdiene til skjerm
eller webside.

```
let biler = [];
        biler.push(new Kjoretoy(10000, 210));

        for(let bil in biler){
            let print = document.createElement("div");
            print.innerHTML = `
              Farge: ${kjoretoy.farge} <br>
              Kjørelengde: ${kjoretoy.kjorelengde} <br>
              Makshastighet: ${kjoretoy.makshastighet}
            `;
            //print.innerHTML = JSON.stringify(biler);
            document.body.appendChild(print);
        }
```

c. *Lag kode for klassen Buss. Klassen skal arve alle variabler og metoder fra moderklassen*
Kjoretoy. Klassen skal ha en variabel makspassasjerer i tillegg til de som blir arvet. Denne
variabelen forteller hvor mange passasjerer det er plass til i bussen.

```
class Buss extends Kjoretoy{
            constructor(kjorelengde, makshastighet, makspassasjer){
                super(kjorelengde, makshastighet);
                this.makspassasjer = makspassasjer;
            }
        }
```

d. *Lag en metode sjekkAntall i klassen Buss. Metoden skal ha argument antallpassasjerer,*
sjekker så om antallet passasjerer overstiger makspassasjerer og returnerer en boolsk
verdi.

```
sjekkAnt(antallPassasjer){
                if(antallPassasjerer > this.makspassasjerer) return false;
                else return true;

                //evt: return antallPassasjerer <= this.makspassasjer;
            }
```

e. *Lag kode som oppretter en buss som en instans av klassen Buss. Velg selv navn på dette*
objektet og passende verdier for alle variablene. Skriv deretter ut en melding til skjerm eller
webside der du bruker sjekkAntall til å teste om det er plass til 50 passasjerer i bussen.

```
let buss = new Buss(8000, 100, 60);
        console.log(buss.sjekkAnt(50));
```

f. *Legg til en variabel i koden som gjør at alle kjøretøy, både vanlige og busser, har fargen hvit.*

```
//legger til i klassen fra oppgave a:
class Kjoretoy{
```

```
            farge = "hvit";
            constructor(kjorelengde, makshastighet){
             this.makshastighet = makshastighet;
             this.kjorelengde = kjorelengde;
            }
        }

console.log(buss.farge); //output = "hvit"
```

g. *Lag en metode med valgfritt navn i klassen Buss som beregner hvor mye det koster å leie*
hele bussen for en dag. Prisen er avhengig størrelsen på bussen (makspassasjerer). Hvis
*det er 50 seter i bussen blir prisen 50*100 = 5000 kr. I tillegg legges det til moms på 25% slik*
at totalprisen blir 7500 kr. Bruk objektet du opprettet tidligere og skriv ut totalprisen til skjerm
eller webside.

```
dagsLeie(){
                let mva = 0.25;
                let leiePrPers = 100;
                let prisUmva = this.makspassasjerer * leiePrPers;
                let totalSum = prisUmva * (1+mva)
                return totalSum;
            }
console.log(buss.dagsLeie);
```

Hele koden med typesjekking (tsx)

```
class Kjoretoy{
    makshastighet: Number;
    kjorelengde: Number;
    farge: string = "hvit";
    constructor(kjorelengde: Number, makshastighet: Number){
        this.makshastighet = makshastighet;
        this.kjorelengde = kjorelengde;
    }
}

let kjoretoy = new Kjoretoy(10000, 210);
console.log(kjoretoy);

class Buss extends Kjoretoy{
    makspassasjerer: number;
    constructor(kjorelengde: Number, makshastighet: Number, makspassasjerer: number){
```

```
        super(kjorelengde, makshastighet);
        this.makspassasjerer = makspassasjerer;
    }

    sjekkAnt(antallPassasjerer: number){
        if(antallPassasjerer > this.makspassasjerer) return false;
        else return true;

        //evt: return antallPassasjerer <= this.makspassasjer;
    }

    dagsLeie(){
        let mva = 0.25;
        let leiePrPers = 100;
        let prisUmva = this.makspassasjerer * leiePrPers;
        let totalSum: Number = prisUmva * (1+mva)
        return totalSum;
    }
}

let buss = new Buss(8000, 100, 60);
console.log(buss.sjekkAnt(50));
console.log(buss.farge)
console.log(buss.dagsLeie());
```

OBS! Det er forskjell på Number og number (stor og liten “n”). Merk at for å kunne bruke makspassesjerer i multiplikasjon så må den ellers være definert med liten bokstav, ellers får man feilmelding.

Oppgave 2

a. Forklar begrepene objekt, klasse og instans.

b. Hva gjør funksjonen super() når den blir kalt i en konstruktør?

c. Hva er grunnen til at det vanligvis brukes unntakshåndtering når vi kobler oss til en database?

d. Det kan være lurt å legge kode som hører til databaseoppkoblingen i en egen klasse. Hvorfor er det slik?

e. Hvis du skulle lagre påloggingsinformasjonen til databasen i programmet ditt, hvordan ville du gjort dette?

Objekt, klasse og instans i henhold til objektorientert programmering:

- Objekt: En konkret, selvstendig enhet som har egenskaper og handlinger. Objekter kan være f.eks. biler, bøker eller personer.
- Klasse: En abstrakt beskrivelse av en type objekter, som angir hvilke egenskaper og handlinger objektene av denne klassen har. Klassen fungerer som en mal for objekter.
- Instans: Et objekt som er laget på bakgrunn av en klasse. Instanser har de samme egenskapene og handlingene som klassen, men med konkrete verdier.

I eksempelet fra oppgave 1 er Kjoretoy en klasse som beskriver egenskapene og handlingene til et kjøretøy. Et objekt av klassen Kjoretoy er en konkret instans av et kjøretøy, med konkrete egenskaper som makshastighet og kjørelengde. Buss-klassen er en underklasse av Kjoretoy-klassen, og beskriver egenskapene og handlingene til en spesifikk type kjøretøy. Et objekt av typen Buss er en instans av klassen Buss, og har egenskaper som makspassasjerer i tillegg til egenskapene som arves fra Kjoretoy-klassen.

Super()

super() er en funksjon som blir kalt i en konstruktør i en underklasse, og som refererer til konstruktøren i superklassen. Ved å kalle super() blir egenskapene og metodene fra superklassen arvet av underklassen, og kan dermed brukes i underklassens konstruktør. Dette gjør det mulig å lage en underklasse som har tilleggsegenskaper og -metoder i tillegg til de som er arvet fra superklassen.

Unntakshåndtering

Unntakshåndtering brukes vanligvis når man kobler seg til en database for å fange opp eventuelle feil eller unntak som kan oppstå under tilkoblingen. Dette kan for eksempel være hvis databasen midlertidig er nede eller hvis det er en feil i tilkoblingsinformasjonen. Ved å bruke unntakshåndtering kan man håndtere slike situasjoner på en mer elegant og kontrollert måte, og gi tilbakemelding til brukeren om hva som har skjedd og eventuelle tiltak som kan tas for å rette opp feilen.

Unntakshåndtering (engelsk: exception handling) er en teknikk innen programmering som brukes for å fange opp og håndtere feil eller unntak som kan oppstå under utførelsen av et program. Hensikten med unntakshåndtering er å sørge for at programmet ikke krasjer eller stopper uventet når det oppstår en feil,

og å gi tilbakemelding til brukeren om hva som har skjedd og eventuelle tiltak som kan tas for å rette opp feilen. Unntakshåndtering gjør det også mulig å skille mellom ulike typer feil og å håndtere dem på forskjellige måter, avhengig av hva som er hensiktsmessig i den aktuelle situasjonen.

Et eksempel på en objekt kan være en bil, som har egenskaper som merke, modell og farge, og metoder som akselerer og brems. En klasse ville være en beskrivelse av en type objekt, som for eksempel en Bil-klasse, som ville definere egenskapene og metodene som alle bilobjekter ville ha. En instans ville være en spesifikk bil, for eksempel en rød 2018 Toyota Camry, som ville være en instans av Bil-klassen og ville ha alle egenskapene og metodene definert av klassen.

Database i egen klasse

Det kan være lurt å legge kode som hører til databaseoppkoblingen i en egen klasse av flere grunner. For det første kan det gjøre koden mer oversiktlig og enkel å vedlikeholde, da all kode som har med databaseoppkobling å gjøre ligger samlet på ett sted. På denne måten kan man også lettere gjenbruke kode som har med databaseoppkobling å gjøre i andre deler av programmet. En annen fordel med å legge databaseoppkoblingskoden i en egen klasse er at det gjør det enklere å implementere ulike typer databaser eller tilkoblingsmetoder, da man kan bytte ut hele klassen eller enkelte deler av den uten å måtte endre på resten av koden. Til slutt kan det også øke sikkerheten, da man kan legge til ekstra sjekker og godkjenninger når man kobler seg til databasen i en egen klasse.

Lagring av passord

Hvis jeg skulle lagre påloggingsinformasjonen til databasen i programmet mitt, ville jeg først og fremst sørget for å lagre informasjonen på en sikker måte, for eksempel ved å kryptere passordet. Deretter ville jeg lagret informasjonen i en konfigurasjonsfil eller i en miljøvariabel, slik at den enkelt kan endres eller oppdateres uten å måtte endre koden. Jeg ville også sørget for å begrense tilgangen til denne informasjonen så mye som mulig, for eksempel ved å legge tilgodkjenninger eller autorisasjonsmekanismer for å sikre at bare autoriserte brukere har tilgang til informasjonen.

Oppgave 3

Du skal lage en chat-applikasjon med mulighet for flere rom som brukerne kan sende
meldinger til. Databasen er allerede opprettet med følgende SQL-setninger:

```
CREATE TABLE ChatRooms (
    id INT NOT NULL AUTO_INCREMENT,
    title TEXT,
    description TEXT,
    PRIMARY KEY(id)
);
CREATE TABLE Messages (
    id INT NOT NULL AUTO_INCREMENT,
    text TEXT,
    chatRoomId INT NOT NULL,
    PRIMARY KEY(id)
);
```

Du kan gå ut i fra at databaseoppkoblingen allerede er satt opp.
Applikasjonen skal ha følgende funksjonaliteter:

- Ved oppstart skal en liste over alle chat-rommene vises
- Når en går inn i et chat-rom skal alle meldingene i chat-rommet vises
- Det skal være mulig å legge til et chat rom
- Det skal være mulig å legge til en melding i et chat rom

For å få full uttelling på denne oppgaven må kildekoden i applikasjonen være godt strukturert slik at det er enkelt å utvide applikasjonen senere. Det gis ekstra poeng for å ta i bruk Promiseobjekter og/eller statisk typesjekking.

```
import * as React from 'react';
import { Component } from 'react-simplified';
import ReactDOM from 'react-dom';
import { pool } from './mysql-pool';
import { Card, Row, Column, Button, Form, NavBar } from './widgets';
import { NavLink, HashRouter, Route } from 'react-router-dom';

class Chatroom {
    id: number = 0;
    title: string = "";
    description: string = "";
}

class Message {
    id: number = 0;
    text: string = "";
    chatroomId: number = 0;
```

```
}

class ChatroomService {
    getChatrooms(success: (chatRooms: Chatroom[]) => void){
        pool.query('SELECT * FROM ChatRooms', (error: any, results: any) => {
            if (error) return console.error(error);

            success(results);
        });
    }

    getChatroom(id: number, success: (chatRooms: Chatroom[])=> void){
        pool.query('SELECT * FROM ChatRooms WHERE id=?', [id],
          (error: any, results: any) => {
            if(error) return console.error(error);

            success(results[0]);
        });
    }

    addChatroomMessage(chatRoomId: number, text: string, success: () => void){
        pool.query('INSERT INTO Messages (text, chatRoomId) VALUES (?, ?)',
          [text, chatRoomId], (error: any, results: any) => {
            if(error) return console.error(error);

            success();
        });
    }

    getChatroomMessages(chatRoomId: number, success: (messages: Message[])=> void){
        pool.query('SELECT * FROM Messages WHERE chatRoomId =?',
          [chatRoomId], (error: any, results: any) => {
            if(error) return console.error(error);

            success(results);
        });
    }

    addChatroom(chatRoom: Chatroom, success: () => void){
        pool.query('INSERT INTO Chatrooms (title, description) VALUES (?, ?)',
          [chatRoom.title, chatRoom.description], (error: any, results: any) => {
            if(error) return console.error(error);

            success();
        });
    }
}

export let chatroomService = new ChatroomService();
const history = createHashHistory();


class Menu extends Component{
    render(){
        return <NavBar brand="ChatNav">
            <NavBar.Link exact to="/" activeStyle={{ color: 'darkblue' }}>
                ChatAdm
            </NavBar.Link>{' '}
```

```
            <NavBar.Link to="/chats" activeStyle={{ color: 'darkblue' }}>
                Chats
            </NavBar.Link>
        </NavBar>
    }
}

class ChatRooms extends Component{
    chatRooms: Chatroom[] = [];
    newChatRoom = new Chatroom();

    render() {
      return (
        <div>
          <Card title="Chat rooms">
            {this.chatRooms.map((chatRoom) => (
              <Row key={chatRoom.id}>
                <Column>
                  <NavLink to={'/chatRoom/' + chatRoom.id}>{chatRoom.title}</NavLink>
                </Column>
              </Row>
            ))}
          </Card>
          <Card title="New chat room">
            <Form.Label>Title</Form.Label>
            <Form.Input
              type="text"
              value={this.newChatRoom.title}
              onChange={(event: { currentTarget: { value: any; }; }) =>
              (this.newChatRoom.title = event.currentTarget.value)}
            />
            <Form.Label>Description</Form.Label>
            <Form.TextArea
              value={this.newChatRoom.description}
              onChange={(event: { currentTarget: { value: any; }; }) =>
              (this.newChatRoom.description = event.currentTarget.value)}
            />
            <Button.Success
              onClick={() =>
                chatroomService.addChatroom(this.newChatRoom, () => {
                  this.newChatRoom = new Chatroom();
                  this.mounted();
                })
              }
            >
              Create chat room
            </Button.Success>
          </Card>
        </div>
      );
    }

    mounted() {
      chatroomService.getChatrooms((chatRooms) => (this.chatRooms = chatRooms));
    }
}

class ChatRoomDetails extends Component<{ match: { params: { id: string } } }>{
```

```
    chatRoom = new Chatroom();
    messages: Message[] = [];
    newMessage = new Message();
    props: any;

    render() {
        return (
            <div>
                <Card title="Chat room">
                    <Card title={this.chatRoom.title}>
                        {this.chatRoom.description}
                        <Card title="Messages">
                            {this.messages.map((message) => (
                                <Row key={message.id}>
                                    <Column>{'> ' + message.text}</Column>
                                </Row>
                            ))}
                        </Card>
                    </Card>
                    <Card title="New message">
                        <Form.TextArea
                            value={this.newMessage.text}
                            onChange={(event: { currentTarget: { value: string; }; })
                             => (this.newMessage.text = event.currentTarget.value)}
                        />
                        <Button.Success
                            onClick={() =>
                                chatroomService.addChatroomMessage(
                                Number(this.props.match.params.id),
                                    this.newMessage.text,() => {
                                        this.newMessage = new Message();
                                        this.mounted();
                                    }
                                )
                            }
                        >
                            Create message
                        </Button.Success>
                    </Card>
                </Card>
            </div>
        );
    }

    mounted() {
        chatroomService.getChatroom(
            Number(this.props.match.params.id),
            (chatRoom) => (this.chatRoom = chatRoom)
        );
        chatroomService.getChatroomMessages(
            Number(this.props.match.params.id),
            (messages) => (this.messages = messages)
        );
    }
}

ReactDOM.render(
    <HashRouter>
```

```
      <Menu />
      <Route exact path="/" component={ChatRooms} />
      <Route exact path="/chatRoom/:id" component={ChatRoomDetails} />
    </HashRouter>,
    document.getElementById('root')
  );
```