2020

Oppgave 1

a. Lag JavaScript-kode for klassen Sirkel.
   Klassen Sirkel skal inneholde to private variabler - radius og farge.
   Defaultverdien for radius er 1 og defaultverdien for farge er "red".
   Klassen skal ha to konstruktører - en som ikke tar argumenter og en som tar en valgfri radius større enn 0 som argument.
   Klassen skal ha tre offentlige metoder - en som returnerer radiusen til sirkelen, en som returnerer arealet til sirkelen og en som returnerer omkretsen til sirkelen.

```
//oppgave a
class Sirkel{
    private _radius?: number = 1;
    private _color?: string = 'red';

    constructor(radius: number, color: string){
        if (radius && radius > 0) this._radius = radius;
        if (color) this._color = color;

    }

    get radius() {return this._radius};
    get color() {return this._color};

    set radius(value: number|undefined) {this._radius = value};
    set color(value: string|undefined) {this._color = value};

    get areal() { if (this._radius) return (Math.PI * this._radius)**2; }
    get omkrets() { if (this._radius) return 2*Math.PI * this._radius; }

    toString() {
        if (this._radius) {
            return(
                `Sirkelen har en radius på ${this._radius},
                  areal på ${this.areal}
                  og omkrets på ${this.omkrets}. `
            );
        } else {
            `Sirkelen mangler radius`;
        }

    }
}
```

b. Lag kode som oppretter to objekter ved hjelp av klassen Sirkel.
   Opprett den første sirkelen med defaultverdier. Den andre sirkelen skal ha radius 2.
   Skriv ut informasjon om de to sirklene til konsoll eller webside, f.eks. "Sirkelen har en radius på 2, et areal på 12,56 og en omkrets på 12,56". Svarene i utskriften bør rundes av til to desimaler.

```
//Oppgave b
const opprettSirkel = new Sirkel();
const nySirkel = new Sirkel(2, "blue");
console.log(opprettSirkel.toString());
console.log(nySirkel.toString());
```

c. Lag JavaScript-kode for klassen Kube.
   Klassen Kube skal inneholde en privat variabel - side.
   Klassen skal ha en konstruktør som tar et sirkelobjekt som argument.

```
//Oppgave c
class Kube {
    private _side: number;
    private _sirkel: Sirkel;

    constructor(side: number, sirkel: Sirkel){
        this._side = side;
        this._sirkel = sirkel;
    }
}

const kube = new Kube(2, nySirkel);
```

   Det vi gjør her er å bruke objektet nySirkel som argument når vi lager en kube. Dette betyr at vi essensielt sett ville fått en kube basert på målene til den nye sirkelen.

Oppgave 2

Sjakk er et spill der vi har et spillebrett med 8x8 ruter og starter med 32 spillebrikker; 16 bønder, 2 konger, 2 dronninger, 4 tårn, 4 løpere og 4 springere.

Lag klasser for spillebrikkene. La de klassene du lager arve fra en superklasse.

Hver brikke kan være enten svart eller hvit. Hver brikke har en posisjon X = A-H og Y = 1-8 på
spillebrettet. Disse må kunne lagres som variabler.

Lag kode som oppretter alle spillebrikkene som objekter med riktig farge og startposisjon.

```
abstract class SpilleBrikke {
    protected _name: string = "brikke";
    private _color: string = '';
    private _xPos: string = '';
    private _yPos: number;
    private _validX: string[] = ["A", "B", "C", "D", "E", "F", "G", "H"];
    private _validY: number[] = [1, 2, 3, 4, 5, 6, 7, 8];

    constructor(color: string, xPos: string, yPos: number) {
        this._color = color;

        if (this.validateX(xPos)) this._xPos = xPos;
        else throw new Error('Posisjonen til X må være en bokstav mellom A og H');

        if (this.validateY(yPos)) this._yPos = yPos;
        else throw new Error('Posisjonen til y må være et tall mellom 1 og 8');
    }

    get color() { return this._color; }
    get xPos() { return this._xPos; }
    get yPos() { return this._yPos; }

    validateX(pos: string): boolean {
        for (const letter of this._validX) if (pos.toUpperCase() === letter) return true;
        return false;
    }

    validateY(pos: number): boolean {
        for (const number of this._validY) if (pos === number) return true;
        return false;
    }

    toString(){
        return(
            `${this._color} ${this._name} har posisjon ${this._xPos}${this._yPos}`
        );
    }
}

//lager klasser for spillebrikkene som arver fra superklassen.
class Bonde extends SpilleBrikke{
    _name = "Bonde";

    constructor(color: string, xPos: string, yPos: number){
        super(color, xPos, yPos);
    }
}
```

```
class Taarn extends SpilleBrikke{
    _name = "Tårn";

    constructor(color: string, xPos: string, yPos: number){
        super(color, xPos, yPos);
    }
}

class Loper extends SpilleBrikke{
    _name = "Løper";

    constructor(color: string, xPos: string, yPos: number){
        super(color, xPos, yPos);
    }
}

class Springer extends SpilleBrikke{
    _name = "Springer";

    constructor(color: string, xPos: string, yPos: number){
        super(color, xPos, yPos);
    }
}

class Konge extends SpilleBrikke{
    _name = "Konge";

    constructor(color: string, xPos: string, yPos: number){
        super(color, xPos, yPos);
    }
}

class Dronning extends SpilleBrikke{
    _name = "Dronning";

    constructor(color: string, xPos: string, yPos: number){
        super(color, xPos, yPos);
    }
}

//oppretter alle brikkene på brettet med riktig farge og posisjon
const tårn: Taarn[] =
  [new Taarn("white", "A", 1),
  new Taarn("white", "H", 1),
  new Taarn("black", "H", 8),
  new Taarn("black", "A", 8)];

const springere: Springer[] =
  [new Springer("white", "B", 1),
  new Springer("white", "G", 1),
  new Springer("black", "B", 8),
  new Springer("black", "G", 8)]

const løpere: Loper[] =
  [new Loper("white", "C", 1),
  new Loper("white", "F", 1),
  new Loper("black", "F", 8),
  new Loper("black", "C", 8)];
```

```
const konger: Konge[] =
  [new Konge("white", "E", 1),
  new Konge("black", "E", 8)];

const dronninger: Dronning[] =
  [new Dronning("white", "D", 1),
  new Dronning("black", "D", 8)];

const bønder: Bonde[] = [];
const letters: string[] = ["A", "B", "C", "D", "E", "F", "G", "H"];
/*her lager vi en array for bokstavene slik at vi kan kjøre gjennom dem
som løkke og plassere alle bøndene i samme funksjon, se neste linje*/
for (let i = 0; i < 8; i++) bønder.push(new Bonde("white", letters[i], 2));
for (let i = 0; i < 8; i++) bønder.push(new Bonde("black", letters[i], 7));

for (const bonde of bønder) console.log(bonde.toString());
for (const t of tårn) console.log(t.toString());
for (const springer of springere) console.log(springer.toString());
for (const løper of løpere) console.log(løper.toString());
for (const konge of konger) console.log(konge.toString());
for (const dronning of dronninger) console.log(dronning.toString());
```

Oppgave 3

Du skal lage en applikasjon for NRK som viser Barne-TV programmer lagret i en database. Databasen er allerede opprettet med følgende SQL-setninger:

```
CREATE TABLE Shows (
    id INT NOT NULL AUTO_INCREMENT,
    title TEXT,
    description TEXT,
    PRIMARY KEY(id)
);
CREATE TABLE ShowRatings (
    id INT NOT NULL AUTO_INCREMENT,
    rating INT NOT NULL,
    showId INT NOT NULL,
    PRIMARY KEY(id)
);
```

Applikasjonen skal ha følgende funksjonaliteter:

- Ved oppstart skal alle Barne-TV programmene vises med tittel og beskrivelse
- Det skal være mulig å legge til et Barne-TV program
- Det skal være mulig å gi terningkast på et Barne-TV program
- Gjennomsnittet av gitte terningkast skal vises for hvert Barne-TV program

- Det skal være mulig å søke etter et Barne-TV program

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 Promise-objekter og/eller statisk typesjekking.

Index.tsx

```
import * as React from 'react';
import { createRoot } from 'react-dom/client';
import { Component } from 'react-simplified';
import { NavLink, HashRouter, Route } from 'react-router-dom';
import { Shows, ShowRating, showService } from './services';
import { Alert, Card, Row, Column, NavBar, Button, Form } from './widgets';
import { createHashHistory } from 'history';

const history = createHashHistory();

class Menu extends Component {
    render() {
        return (
            <div>
                <NavBar>
                    <NavLink exact to ="/" activeStyle={{ color: 'darkblue' }}>
                      Shows
                    </NavLink>{ ' ' }
                    <NavLink exact to ="/newShow" activeStyle={{ color: 'darkblue' }}>
                      New show
                    </NavLink>{ ' ' }
                </NavBar>
            </div>
        )
    }
}

class Home extends Component {
  // Definerer to variabler som er tilgjengelige på instansen av komponenten
  searchInput: string = '';
  shows: Shows[] = [];

  render(){
    // Returnerer JSX som definerer hvordan komponenten skal se ut
    return(
      <div>
        <h1>List of all programs: </h1>

        <Row>
          <Card title='BarneTV-programmer'>
            <Form.Input
              type='text'
              value={this.searchInput}
              onChange={(e: { target: { value: string; }; }) =>
                        this.searchInput = e.target.value}
              required={true}
```

```
              placeholder='Search'
              //Definerer et tekstfelt som brukeren kan skrive inn i,
                  //og to knapper for å søke og nullstille
            />
            <Button.Success
              onClick={() => this.search()} //Kobler knappen til search-metoden
                                            //når den trykkes på
            >
              Search
            </Button.Success>
            <Button.Light
              onClick={() => this.reset()} // Kobler knappen til reset-metoden
                                          //når den trykkes på
            >
              Reset
            </Button.Light>
          </Card>

          {
            // Mapper gjennom listen over shows og lager en tabellrad for hver
            this.shows?.map((show, index) => {
              return <ShowCard key={index} show={show} />
            })
            //vi mapper gjennom index også, fordi hvert "child" i en liste
              //bør ha en unik nøkkel
          }

          <Button.Light
            onClick={() => history.push("/newShow")}
            // Navigerer til en annen side når knappen trykkes på
          >
            Add a new show
          </Button.Light>
        </Row>
      </div>
    );
  }

  reset(){
// Henter alle shows fra databasen og oppdaterer instansvariabelen shows med resultatene
    showService.getShows((shows) => {
      this.shows = shows;
    })
  }

  search(){
    // Henter shows fra databasen som har tittelen som brukeren har søkt etter
    showService.getShowsByTitle(this.searchInput, (shows) => {
      // Oppdaterer instansvariabelen shows med resultatene
      this.shows = shows;
    });
  }

  mounted(){
// Henter alle shows fra databasen og oppdaterer instansvariabelen shows med resultatene
    showService.getShows((shows) => {
      this.shows = shows;
    });
```

```
  }
}


class ShowCard extends Component {
  props: any;
  showRating: ShowRating[] = [];

  render() {
    const avgRating = this.showRating.length > 0
      ? (this.showRating.reduce((a: any, b: any) => a + b.rating, 0)
          / this.showRating.length).toFixed(2)

      : 'Ingen vurderinger';

      /*sjekker om array til rating er større enn null og returnerer gjennomsnitt,
      hvis lengden er null returnerer den "ingen vurderinger" */

    return (
      <Card title={this.props.show.title}>
        <Row>{this.props.show.description}</Row>
        <Row>Average rating: {avgRating}</Row>
        <NavLink to={`/rating/${this.props.show.id}`}>
          Rate the show
        </NavLink>
      </Card>
    );
  }

  mounted() {
    showService.getRatings(this.props.show.id, (ratings: ShowRating[]) => {
      this.showRating = ratings;
    });
  }
}

class NewRating extends Component<{ match: { params: { id: string } } }> {
  props: any;

  // Lager et tomt rating-felt med default verdi 0
  rating: number = 0;

// Oppretter nytt objekt av Shows-klassen som brukes til å hente ut informasjon om showet
  show = new Shows();

  render() {
    return (
      <Card title={`New rating of ${this.show.title}`}>
        <Row>
          <Form.Label>Rating</Form.Label>

          {/* Input felt for å legge til rating */}
          <Form.Input
            type="text"
            value={this.rating}
            onChange={(e: { target: { value: any } }) =>
              (this.rating = Number(e.target.value))
            }
```

```
          />
        </Row>

        <br />

        {/* Knapp for å lagre rating */}
        <Button.Success onClick={() => this.add()}>Save</Button.Success>

        {/* Knapp for å avbryte og gå tilbake til forsiden */}
        <Button.Danger onClick={() => history.push("/")}>
          Cancel
        </Button.Danger>
      </Card>
    );
  }

  // Henter ut showet med gitt ID når komponenten er montert
  mounted() {
    showService.getShow(Number(this.props.match.params.id), (show) => {
      this.show = show;
    });
  }

  // Legger til ratingen til showet
  add() {
    // Sjekker om ratingen er innenfor gyldig område (1-6)
    if (this.rating < 1 || this.rating > 6) return;

    // Legger til ratingen til showet
    showService.addNewRating(
      Number(this.props.match.params.id),
      Number(this.rating)
    );

    // Går tilbake til forsiden
    history.push("/");
  }
}

class NewShow extends Component {
  // Oppretter et tomt show-objekt
  show = new Shows();

  render() {
    return (
      <Card>
        {/* Input felt for å legge til tittel */}
        <Row>
          <Form.Label>Title</Form.Label>
          <Form.Input
            type="text"
            value={this.show.title}
            onChange={(e: { target: { value: string | undefined } } ) =>
              (this.show.title = e.target.value)
            }
          />
        </Row>
```

```
        {/* Input felt for å legge til beskrivelse */}
        <Row>
          <Form.Label>Description</Form.Label>
          <Form.Input
            type="text"
            value={this.show.description}
            onChange={(e: { target: { value: string | undefined } } ) =>
              (this.show.description = e.target.value)
            }
          />
        </Row>

        <br />

        {/* Knapp for å lagre showet */}
        <Button.Success onClick={() => this.add()}>Save</Button.Success>

        {/* Knapp for å avbryte og gå tilbake til forsiden */}
        <Button.Danger onClick={() => history.push("/")}>
          Cancel
        </Button.Danger>
      </Card>
    );
  }

  // Legger til showet
  add() {
    showService.addNewShow(this.show);

    // Går tilbake til forsiden
    history.push("/");
  }
}

// Setter opp ruting for forskjellige deler av applikasjonen
let root = document.getElementById('root');
if (root)
    createRoot(root).render(
        <div>
            <Alert />
            <HashRouter>
              <Menu />
                <div>
                    <Route exact path="/" component={Home} />
                    <Route exact path="/newshow" component={NewShow} />
                    <Route exact path="/rating/:id" component={NewRating} />
                </div>
            </HashRouter>
        </div>
    )
```

Services.tsx

```
import { pool } from './mysql-pool';
import type { RowDataPacket, ResultSetHeader } from 'mysql2';

// Definerer klassen Shows med tre attributter
export class Shows {
  id: number | undefined;
  title: string | undefined;
  description: string | undefined;
}

// Definerer klassen ShowRating med tre attributter
export class ShowRating {
  id: number | undefined;
  rating: number | undefined;
  showId: number | undefined;
}

// Definerer klassen ShowService med to funksjoner for å hente ut
//informasjon om TV-serier
class ShowService {

  // Funksjon for å hente ut alle TV-serier fra databasen
  getShows(success: (shows: Shows[]) => void){
      pool.query(
      'SELECT * FROM Shows ', (error: any, results: any) => {
          if (error) return console.error(error);

          // Kaller success-funksjonen med resultatene fra spørringen
          success(results);
      });
  }

  // Funksjon for å hente ut informasjon om en spesifikk TV-serie basert på ID
  getShow(id: number, success: (show: Shows) => void){
    pool.query(
      'SELECT * FROM Shows WHERE id=?', [id], (error: any, results: any) => {
        if (error) return console.error(error);

        // Kaller success-funksjonen med informasjonen om TV-serien som ble funnet
        success(results[0]);
      }
    )
  }

  getShowsByTitle(title: string, success: (show: Shows[]) => void){
    pool.query(
      // Spørring for å hente ut alle rader fra Shows-tabellen som
        //matcher tittelen som er gitt som parameter.
      // Vi bruker ?-placeholder for å unngå SQL-injeksjoner,
        //og erstatter det med tittelstrengen som er gitt i et array.
      'SELECT * FROM Shows WHERE title LIKE ?', [`%${title}%`],
      (error: any, results: any) => {
      // Hvis det oppstår en feil, skriver vi ut feilmeldingen og returnerer funksjonen.
        if (error) return console.error(error);

      // Vi filtrerer resultatene for å finne de som faktisk
        //inneholder tittelstrengen (case-insensitive)
```

```
      const filteredShows = results.filter((show: any) => show.title.includes(title));

      // Vi mapper over de filtrerte resultatene for å
        //konvertere dem til instanser av klassen Shows
      const shows = filteredShows.map((show: any) => {
        const s = new Shows();
        s.id = show.id;
        s.title = show.title;
        s.description = show.description;
        return s;
      });

      // Når vi har fått konvertert alle radene til instanser av Shows-klassen,
      //kaller vi success-funksjonen som vi fikk som parameter,
      // og sender med listen av shows.
      success(shows);
    }
  );
}


addNewShow(show: Shows){
  // Legger til ny TV-serie i databasen
  pool.query(
    'INSERT INTO Shows (title, description) VALUES(?,?)',
      [show.title, show.description], (error: any, results: any) => {
      if (error) return console.error(error);

      return results[0];
    }
  );
}

/*
getRatings(id: number, success: (rating: ShowRating) => void){
    // Funksjon for å hente ut vurderinger for en TV-serie med angitt ID.
    // Men funksjonen er kommentert ut og ikke i bruk for øyeblikket.
    pool.query(
    'SELECT rating FROM ShowRating sr INNER JOIN Shows s ON sr.showId=s.id WHERE id=?',
       [id], (error: any, results: any) => {
            if (error) return console.error(error);

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

getRatings(showId: number, success: (ratings: ShowRating[]) => void) {
  // Henter ut alle vurderingene som er knyttet til en TV-serie med angitt showId.
  pool.query(
    'SELECT * FROM ShowRating WHERE showId = ?', [showId],
    (error: any, results: RowDataPacket[]) => {
      if (error) return console.error(error);

      // Mapper resultatene til et array av ShowRating-objekter
      const ratings = results.map((result: RowDataPacket) => {
        const rating = new ShowRating();
        rating.id = result.id;
```

```
          rating.rating = result.rating;
          rating.showId = result.showId;
          return rating;
        });

        success(ratings);
      }
    );
  }

  addNewRating(showId: number, rating: number){
    // Legger til en ny vurdering for en TV-serie med angitt showId og rating
    pool.query(
    'INSERT INTO ShowRating (rating, showId) VALUES(?,?)',
      [rating, showId], (error: any, results: any) => {
      if (error) return console.error(error);

      return results;
    });
  }


}

export let showService = new ShowService();
```