React Complete Course Schwarzmuller

1. Getting started

2. JavaScript Refresher

**Classes**

Для чего нужен super() ?

Например, есть два класса, у обоих есть constructor. Он служит для того, чтобы у экземпляра появились свойства класса. Методы прописываются ниже в теле класса.

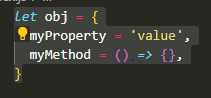
Если мы расширяем первый класс вторым, то, чтобы наследовался конструктор родительского класса, соответственно чтобы появились свойства у экземпляра объекта от первого (наследуемого) класса, используем слово super().

*Если кратко – используем super(), чтобы конструктор родительского класса передался дочернему.*

Именно поэтому в классовых реакт компонентах мы пишем каждый раз super у дочерних компонент.

**Classes, Properties, Methods**

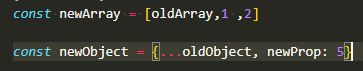
Современный синтаксис по написанию свойств и методов объектов:



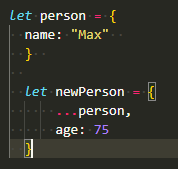
**Spread and rest operators**

Оператор …**spread**:

Копируем данные из **массива**:



Копируем данные из **объекта**:



**Destructuring**

Оператор …**spread**:



2. React Basics &

Working with components

Самая важная концепция в реакте это **компоненты**. В конечном итоге все состоит из них. Любой компонент в Реакте это простая ФУНКЦИЯ!

*Компонент:*

*- может быть переиспользован много раз (DRY)*

*- должен отвечать за что-то одно. Не надо реализовывать много функционала в одном компоненте. (SOLID)*

*Как построен компонент?*

Компонент состоит из: HTML, JS, CSS

Компонент должен быть написан в **декларативном стиле.** Тогда как приложение на обычном JS будет написано по большей части в императивном стиле.

1. The Concept of “Composition”

{props.children}

**Композиция – построение приложения по схеме на картинке. Т.е. каждый дочерний компонент явно указывается в родительском.**

Что, если мы хотим создать компонент – обертку? Для этого применяется **декомпозиция.** Например:

import './Card.css'

//now card is a CONTAINER component

*function* Card (*props*) {

  // так можно применять несколько стилей: стиль обертки и стиль родителя.

  // в строке ПРОБЕЛ!!!

*const* classes = 'card ' + *props*.className

  return <div className={classes}>{*props*.children}</div>

}

export default Card

**И обернем в него 2 компонента:**

import ExpenseItem from './ExpenseItem'

import Card from './Card'

import './Expenses.css'

*function* Expenses ({ *items* }) {

  return (

    <*Card* className={'expenses'}>

      <*ExpenseItem*

        title={*items*[0].title}

        amount={*items*[0].amount}

        date={*items*[0].date}

      />

      <*ExpenseItem*

        title={*items*[1].title}

        amount={*items*[1].amount}

        date={*items*[1].date}

      />

      <*ExpenseItem*

        title={*items*[2].title}

        amount={*items*[2].amount}

        date={*items*[2].date}

      />

      <*ExpenseItem*

        title={*items*[3].title}

        amount={*items*[3].amount}

        date={*items*[3].date}

      />

    </*Card*>

  )

}

**И**

import ExpenseDate from './ExpenseDate'

import './ExpenseItem.css'

import Card from './Card'

*function* ExpenseItem (*props*) {

  return (

    <*Card* className='expense-item'>

      <*ExpenseDate* date={*props*.date} />

      <div className='expense-item\_\_description'>

        <h2>{*props*.title}</h2>

      </div>

      <div className='expense-item\_\_price'>${*props*.amount}</div>

    </*Card*>

  )

}

export default ExpenseItem

**Зачем:**

* **Reusable component с набором стилей и без дочерних компонентов**

1. React State and Events

Если нужно обновлять **стейт с использованием предыдущего состояния** (например, **если** **стейт это объект**), то в функцию, сетающую стейт передать не просто значение, как обычно, а колл-бек:

const titleChangeHandler = (*event*) => {  
 setUserInput((*prevState*)=>({...*prevState*, enteredTitle: *event*.target.value}))  
};

Обычно мы делаем так:

const amountChangeHandler = (*event*) => {  
 setUserInput(*event*.target.value)  
};

Two-way binding

Чтобы не возникало ошибки:

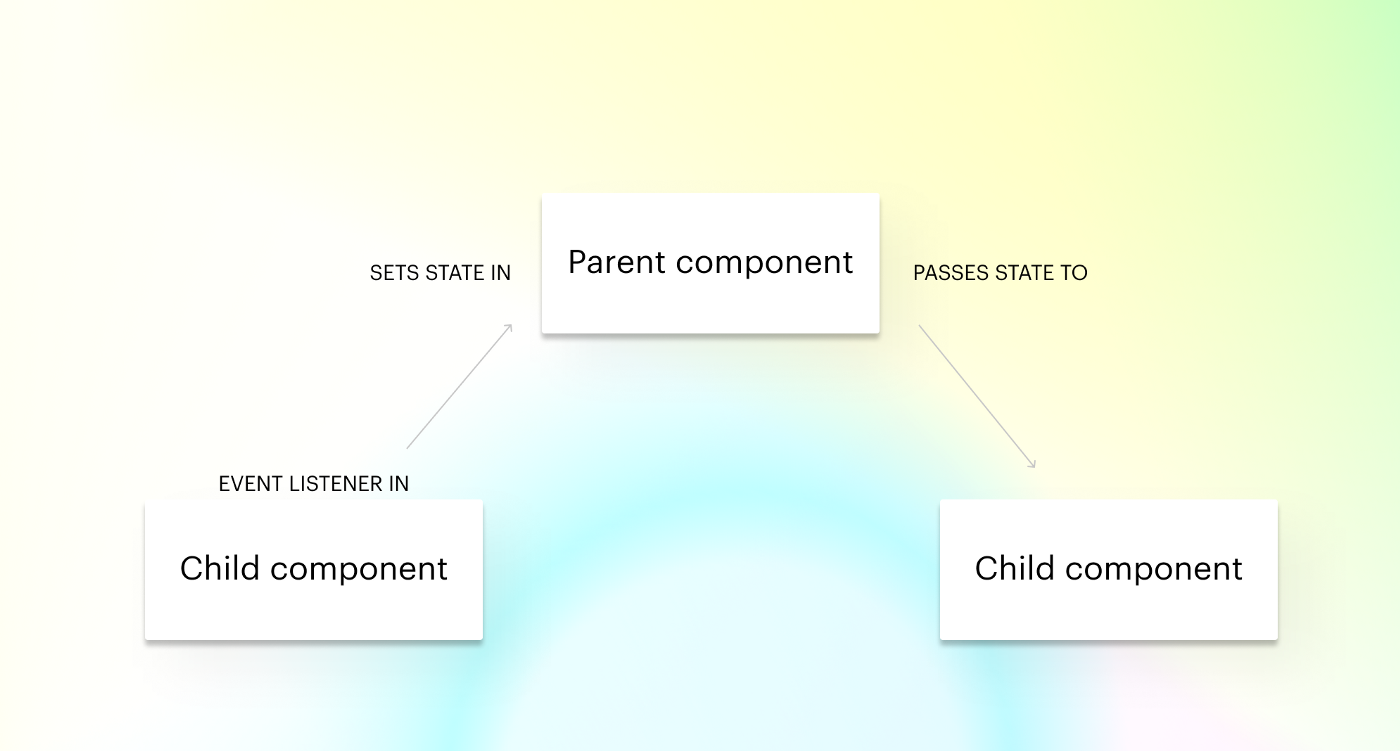
**Warning: A component is changing a controlled input to be uncontrolled. This is likely caused by the value changing from a defined to undefined, which should not happen. Decide between using a controlled or uncontrolled input element for the lifetime of the component.**

Она возникает из-за того, что стейт в первом рендере может быть Undefined.

Нужно прописать в value значение, которое будет отличным от Undefined, а именно:

return (  
 <form onSubmit={submitHandler}>  
 <div className={"new-expense\_\_controls"}>  
 <div className={"new-expense\_\_control"}>  
 <label>**Title**</label>  
 <input type='text' value={userInput.enteredTitle || ''} onChange={titleChangeHandler}/>

Подъем состояния



Данные собираются в дочернем компоненте и через колл-бек возвращаются к родителю. Там они сетаются в стейт и передаются через пропсы во второй дочерний компонент.

Controlled vs Uncontrolled Components

Stateful vs Stateless Components

**Controlled component** – принимает *от родителя* колл-бек через пропсы, передает в него данные, обработанные данные *получает опять от родителя через пропсы*.

**Stateless components** (presentational, dumb) – компонент, у которого нет своего стейта.

1. Rendering Lists & Conditional content

**Rendering Lists** - на входе массив, затем маппинг массива

**Conditional Content** – можем использовать:

1. Тернарные операторы (if else нельзя!)  **«…?...:»**
2. Логические операторы && , ||
3. Вынести JSX разметку в отдельную переменную:

let expensesContent = <p>**No expenses found**</p>;

и использовать ее далее:

function Expenses({*items*}) {  
  
 const [filteredYear, setFilteredYear] = useState('2020');  
  
 const filterChangeHandler = *selectedYear* => {  
 setFilteredYear(*selectedYear*)  
 };  
  
 const filteredExpenses = *items*.filter(*item* => *item*.date.getFullYear() === +filteredYear);  
  
 **let expensesContent = <p>No expenses found</p>;  
  
 if (filteredExpenses.length > 0) {  
 expensesContent = filteredExpenses.map(*expense* => (  
 <ExpenseItem  
 key={*expense*.id}  
 title={*expense*.title}  
 amount={*expense*.amount}  
 date={*expense*.date}/>)  
 )  
 }**  
 return (  
 <div>  
 <Card className={'expenses'}>  
 <ExpensesFilter

selected={filteredYear} onChangeFilter={filterChangeHandler}/>  
 {expensesContent}  
 </Card>  
 </div>  
 )  
}  
  
export default Expenses

1. Сделать **два и более** **отдельных return** в функциональном компоненте. Срабатывает первый, в котором выполняется условие.

if (!showExpenseForm) {

  return (

    <div className={'new-expense\_\_controls'}>

        <button onClick={() *=>* showExpenseFormHandler(true)}>

          Add New Expense

        </button>

    </div>

  )

}

  return (

        <form onSubmit={submitHandler}>

          <div className={'new-expense\_\_controls'}>

            …

          </div>

          <div className={'new-expense\_\_actions'}>

            <button onClick={() *=>* showExpenseFormHandler(false)}>Cancel</button>

            <button type='submit'>Add expense</button>

          </div>

        </form>

  )

}

export default ExpenseForm

Adding Dynamic Styles

Если нужно изменять значение стилей динамически, изменяемую переменную можно передать в **атрибут style**:

*const* ChartBar = *props* *=>* {

*let* barFillHeight = '0%'

  if (*props*.max > 0) {

    barFillHeight = Math.round((*props*.value / *props*.maxValue) \* 100) + '%'

  }

  return (

    <div className={'chart-bar'}>

      <div className={'chart-bar\_\_inner'}>

        <div className={'chart-bar\_\_fill'} style={{height: barFillHeight}}></div>

      </div>

      <div className={'chart-bar\_\_label'}>{*props*.label}</div>

    </div>

  )

}

export default ChartBar