Haushaltshilfe : Data Structures
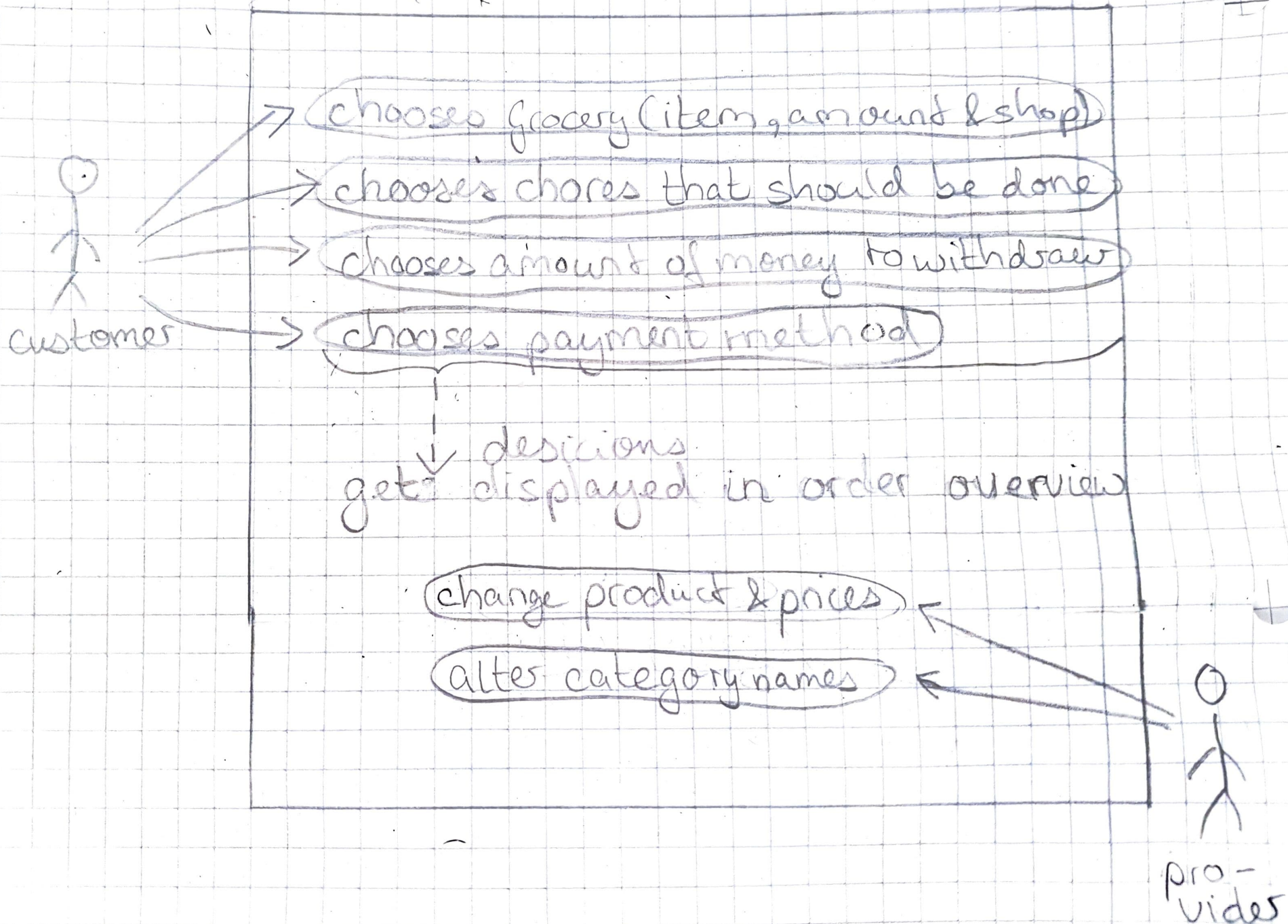
category	name: string	price: number
article	Obst	2,50
	Gemüse	4,00
	Brot	3,20
chores	Sweeping	15
	Dusting	11
payment options	visa	
	Paypal	
	Maestrocard	

data

category

item

Use-Case   Haushaltshilfe

customer

- chooses Grocery (item, amount & shop)
- chooses chores that should be done
- chooses amount of money to withdraw
- chooses payment method

↓ desicions
gets displayed in order overview

- change product & prices
- alter category names

pro-vides

Haushaltshilfe

‹p› — Find sb. to help you with your groceries, your household duties, etc....

‹h1› — Things I want to get help with:
(Choose)

‹select›
name: unit

‹select›
name: shop

‹select›
name: article

• Groceries:

Article | Amount | Unit ▼ | shop ▼ | add ⊕
choose article ▼ | 0   20 |

‹input›
type: range
name: amount

‹button›
id: add
‹click›

‹div›
id: chores
‹change›

input
type: nbr
id: withdraw

household duties
sweeping ☐ laundry ☐
vacuuming ☐ dusting ☐

Manage bank account
withdraw [   ] €

choose paying option:

O Visa
O Mastercard
O PayPal
O

input
type: radio
name: paying
option
‹change›

input

‹div›
id= bank account
‹input›

h1

My order:
• Grocery

article	amount	unit	price	shop	del
					⊖
					⊖
					⊖
					⊖

‹button›
id: delete ?
id: delete

total all articles:	
fee	
total	

• household

chore	price
total:	

Pay with [   ]

• bank account
money withdrawn
money deposited
fees
total

Sum
total order: [   ]

submit
order

‹button›
‹click›

install load listener

- click on add grocery → **handleChangeGrocery**
- change on chores → **handleChange Chores**
- input on banking → **handleChange Banking ...**
- click on submit → **Submit Order**
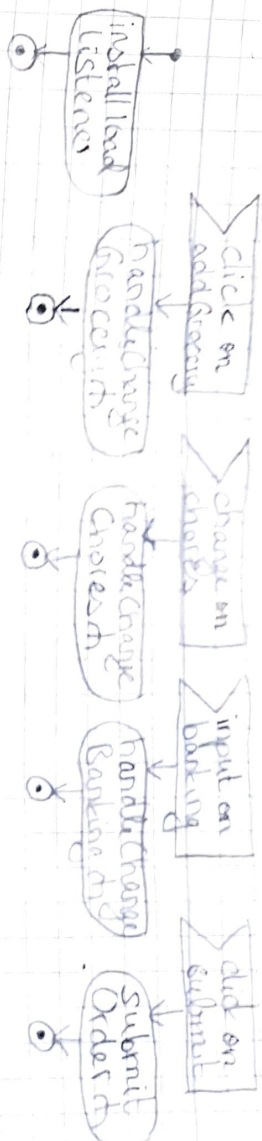
handleLoad → generate Content
- install listeners on +add get Elements
- click: install listeners on Submit button

handleChange Chores event: Event
- let display chores = div#displayChores
- let input = input.checkbox
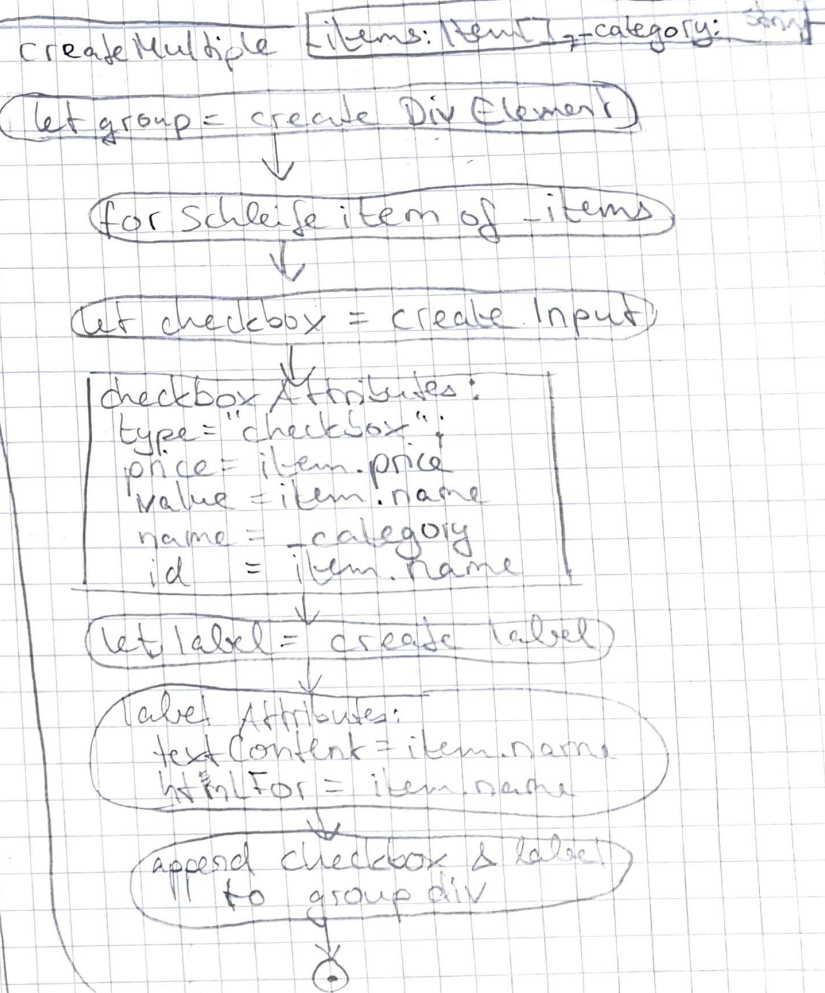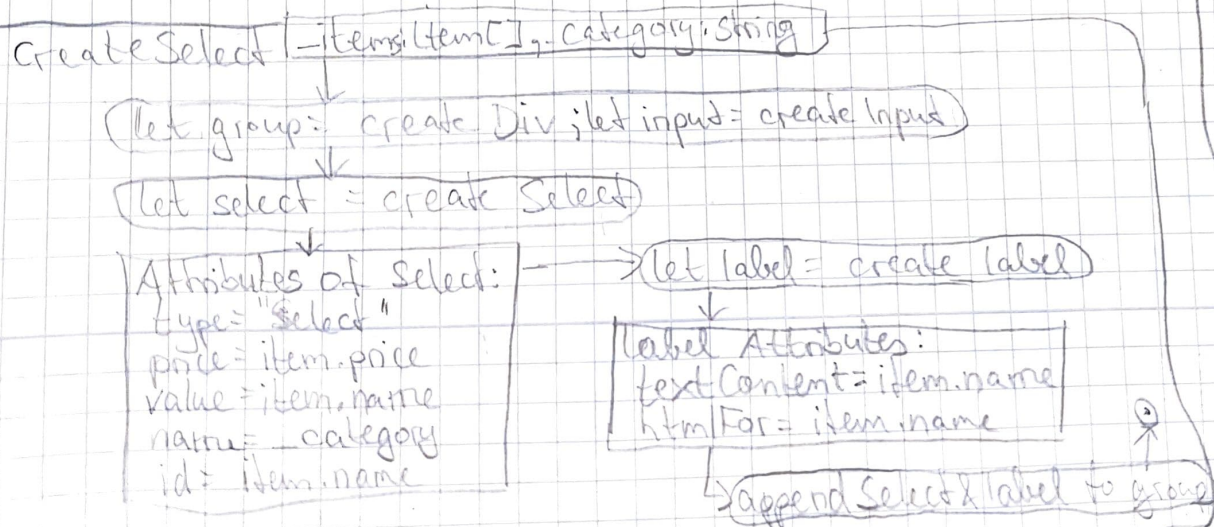- Show all checked inputs in displayChores

handleChangeBanking event: input
- let target = event.target
- let displayBanking: div#displayBanking
- Show target.value in displayBanking

handleChange Grocery
- let article = select #article
- let unit = select #unit
- let shop = select #shop
- let amount = input #amount
- let displayGroceries = div# display Groceries
- let fd = new FormData
- fd.append value of article, amount, unit, shop
- display value of article, unit, shop, price, amount in displayGroceries

SubmitOrder
- alert: new Bestelling ... + fd.append...

Item
name: string
price: number

Data
[category: string]: Item[]

createMultiple (items: Item[], category: item)
- let group = create Div Element
- for Schleife item of items
- let checkbox = create Input
- checkbox Attributes:
  - type = "checkbox";
  - price = item.price
  - value = item.name
  - name = category
  - id = item.name
- let label = create label
- label Attributes:
  - textContent = item.name
  - htmlFor = item.name
- append checkbox & label to group div
- ⊙

generateContent (data: Data)
- let group = null
- [no category attributes] → ⊙
- ◇
  - [category = article] → group = create Select m
  - [category = choices] → group = create Multiple m
  - [category = paymentOptions] → group = create Radio m
- Append Child to Fieldset

Create Select (items: Item[], category: string)
- let group = create Div; let input = create Input
- let select = create Select
- Attributes of Select:
  - type = "Select"
  - price = item.price
  - value = item.name
  - name = category
  - id = item.name
- let label = create label
- label Attributes:
  - textContent = item.name
  - htmlFor = item.name
- append Select & label to group
- ⊙

CreateRadio   [_items: Item[] , _category: string]

(let group = create Div)

(loop for item of _items)

(let radio = create radio)

```
give radio Attributes:
  type = "radio"
  price = 0
  value = item.name
  name = _category
  id = item.name
```

(let label = create label)

```
Attribute: textContent = item.name
           htmlfor     = item.name
```

(append radio & label to group)