Қазақстан Республикасы Білім жане Ғылым министрлігі

Әл Фараби атындағы Қазақ Ұлттық Университеті



**Факультеті:** « Физика – техникалық »

**Кафедрасы:** плазма физикасы, нанотехнология, компьютерлік физика

**СӨЖ**

**Тақырыбы:** JavaScript-те объектіге бағытталған бағдарламалаудың негізгі

принциптері

**Орындаған**: Копжасар Н.Е

**Тексерген**: Асқарұлы Ә.

**Алматы 2022ж**

ОББ принципі - күрделі бағдарламаны құрайтын қарапайым тапсырмаларды шешетін объектілердің жүйесін құру. Нысан жеке өзгеретін күйлерден және осы күйлермен жұмыс істейтін функциялардан тұрады.

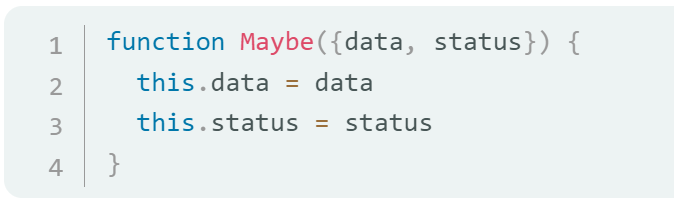
Объектілі - бағытталған бағдарламалау (ОББ) — бұл бағдарламалау парадигмасы, ол абстракция жасау үшін моделі негізделген объектілерінде, нақты әлемде пайдаланады. ОББ бұрын танылған парадигмалармен бірнеше техник пайдаланады, модульдік, полиморфизм және инкапсуляцияны қоса алғанда. Бүгінгі күні көптеген танымал бағдарламалау тілдері (мысалы, Java, JavaScript, C) ОББ-ны қолдайды.

* ***Инкапсуляция***

Инкапсуляция әр объектінің ішкі айнымалыларының сыртқы әсерінен қорғайды. Ең дұрысы, бағдарлама "объектілер аралдарынан" тұруы керек: олардың әрқайсысы өз күйлері бар, хабарламаларды алға - артқа жібереді. Егер сіз тамаша бөлінген жүйені жасасаңыз, бұл жақсы идея сияқты көрінеді, бірақ іс жүзінде мұндай бағдарламаны әзірлеу күрделі және белгілі бір шеңберге енеді.

* ***Полиморфизм***

Полиморфизм деректер түріне қарамастан мінез - құлықты сипаттауға мүмкіндік береді. ОББ бұл басқа деректер түрлерімен жұмыс істейтін объектілер бейімдей алатын класс немесе прототип құруды білдіреді. Полиморфты класс/прототипті қолданатын объектілер жұмыс істеуі үшін деректер түріне тән мінез - құлықты анықтауы керек. Мысалды қарастырайық.



ОББ бағдарламалық қамтамасыз ететін өзара іс - қимыл жасайтын объектілердің жиынтығы ретінде ұсынады, функцияларды немесе қарапайым командаларының тізімінің жиынтығы емес дәстүрлі ұсыну. ОББ әрбір объектіден хабарламалар алады, деректерді өңдеп және басқа да объектілерге хабарламалар жібереді. Әрбір объект кішкентай тәуелсіз машинаның жеке рөлімен немесе жауапкершілігі сияқты ұсынылуы мүмкін.

ОББ бағдарламалауда икемділікті және қолдауды ықпал етеді, және кең таралған кең ауқымды бағдарламалық инжиниринге бөлінген. Өйткені ОББ модульдікті қатаң сынға алады, объектілі - бағытталған кодын әзірлеу оңай және кейіннен оңай түсіну үшін. Кодтау және күрделі жағдайларды түсіну мен рәсімдеріне қарағанда, объектілі - бағытталған коды неғұрлым дәл талдауды ықпал етеді.

* ***Кластар иерархиясы***

Объектілі бағытталған программалау тілі объектілер класс иерархиясының болуын қажет етеді. JavaScript-те мұндай иерархия window объект класынан басталады, яғни әрбір объект белгілі бір терезеге қосылған. Кез келген объектіге немесе қасиетке жүгінгенде бұл объектінің немесе қатынастың берілген объект кіретін иерархияда үлкен болып табылатын объектіден бастап толық немесе жартылай атын көрсету керек:

*window.document.location*

*widow.document.images[0].src*

Негізінен, JavaScript классикалық объектілі тіл болып табылмайды. Онда мұрагерлік пен полиморфизм жоқ. Программист function операторы арқылы өзінің класын анықтай алады, бірақ көптеген жағдайда ол стандартты объектілерді, олардың конструкторларын қолданып, класс деструкторларын мүлдем қолданбайды. Бұл JavaScript – программасының әрекет аумағы ағынды терезеден тыс аумаққа таралмайтындығына байланысты.

Көптеген жағдайда JavaScript-тің әртүрлі объектілерінде бірдей атты қасиеттер анықталады. Бұл жағдайда программист қай объектінің қасиетін қолданғысы келетінін нақты көрсету керек. Мысалы, window пен document-тің location қасиеті бар. Бірақ та window ғана үшін бұл location класс объектісі, ал document үшін – мән ретінде енгізілген құжаттың URL қабылдайтын жол литералы. Көптеген объектілерге оның қасиеттер мәнін қарапайым айнымалыға айналдыратын стандартты әдіс бар екенін ескеру керек. Мысалы, қалыпты жағдайда барлық объектілер үшін белгілер жолына айналу әдісі: toString() анықталған. Location мысалында, егер жолды контекст window.location қолданса, онда қалыпты жағдайға ауысу орындалып, программист оны ескере ала алмайды.

Мысалы:

**class Person {**

**constructor(firstName, lastName) {**

**this.firstName = firstName**

**this.lastName = lastName**

**}**

**getFullName() {**

**return this.firstName + ' ' + this.lastName**

**}**

**}**

Мұнда ES6-дан *class* кілт сөзін қолдана отырып, біз *this-те* сақталатын *firstName* және *lastName* қасиеттері бар жеке класс жасаймыз. Сипат мәндері конструкторда орнатылады және оларға қол жеткізу *getFullName ()* әдісінде жүзеге асырылады.

Біз жаңа кілт сөзді пайдаланып person деп аталатын *person* класының данасын жасаймыз:

**let person = new Person('Dan', 'Abramov')**

**person.getFullName() //> "Dan Abramov"**

**// Мы можем использовать акцессор или получить доступ напрямую**

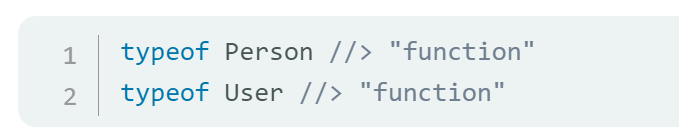
**person.firstName //> "Dan"**

**person.lastName //> "Abramov"**

* ***Прототиптік мұрагерлік***

Прототиптік мұрагерлікте сыныптар мүлдем қолданылмайды. Оның орнына нысандар басқа нысандардан жасалады. Біз жалпыланған объектіден бастаймыз — прототип. Прототипті клондау арқылы басқа объектілерді құру немесе оны әртүрлі функциялармен кеңейту үшін пайдалануға болады.

Алдыңғы тақырыпта біз ES6 - дан сыныпты қалай пайдалану керектігін көрсеткеніммен, JavaScript-тегі сыныптар онша емес:

****

* **Акцессорлар:**

Array.prototype.includes (e) — егер массивте e элементі болса, true қайтарады, әйтпесе - false.

Array.prototype.slice (i, j) — жаңа массивті қайтарады, ол i индексінен j-ге дейінгі бастапқы кесінді болып табылады.

* **Мутаторлар:**

Array.prototype.push (e) - e элементін массивтің соңына орналастырады.

Array.prototype.pop () - массивтің соңғы элементін жояды.

Array.prototype.splice (i, j) — бастапқы кодты сақтамай, i-ден j-ге дейінгі массивтің кесіндісін алады.

Мутаторлар бастапқы массивті өзгертеді. Splice() әдісі slice () сияқты бірдей кесінді шығарады, бірақ егер сізге бастапқы массивті қалдыру қажет болса, онда slice () таңдаған дұрыс.

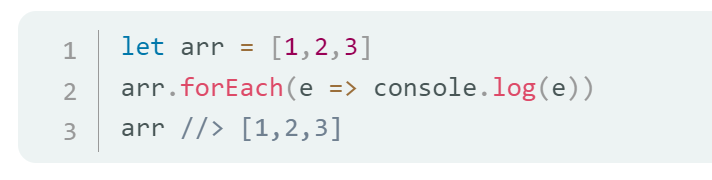
* **Итераторлар:**

Array.prototype.map (f) - массивтің әр элементінде f функциясын қолданады және аталған функцияны шақыру нәтижесі бар жаңа массив жасайды.

Array.prototype.filter (f) - f функциясында берілген тексеруден өткен барлық элементтері бар жаңа массив жасайды.

Array.prototype.forEach (f) - массивтің әр элементінде f функциясын қолданады.

Map() және forEach () әдістері ұқсас, өйткені олар массивтің барлық элементтерімен бірдеңе жасайды, бірақ басты айырмашылық — map() массивті қайтарады, ал forEach () ештеңе жасамайды. Бағдарламалық жасақтаманы жобалаудың жақсы тәжірибелерінде әрқашан жанама әсерлерсіз функцияларды жазу ұсынылады, яғни void функцияларын пайдаланбаңыз. ForEach () әдісі бастапқы массивті өзгертпейді, сондықтан деректерді қандай да бір жолмен түрлендіру қажет болса, map() ең жақсы таңдау болады. ForEach () қолданудың бір мүмкіндігі - күйін келтіру үшін консольге шығару:



***Сыныптар vs прототиптері***

Дэн Абрамов былай дейді:

* Сыныптар JS негізіндегі прототиптік мұрагерлікті жасырады;
* Сабақтар мұрагерлікті қолдануға шақырады, бірақ композицияны қолданған дұрыс;
* Сабақтар, әдетте, сіздің ойыңызға келген алғашқы нашар жоба құрылымын өзгертуге мүмкіндік бермейді.

Сынып иерархиясының орнына бірнеше зауыттық функцияларды жасаған дұрыс. Олар мінез - құлқын реттей отырып, бір - бірін тізбектей шақыра алады. Сондай - ақ, сіз "негізгі" зауыттық функцияны басқа зауыттық функциялардың мінез - құлқын реттейтін "стратегияны" қабылдауға және оны басқа зауыттық функциялардан беруге үйрете аласыз.

***Қорытынды***

Бағдарламашылар көбінесе кодты қайта пайдалану мен оның ауқымдылығы арасындағы масштабын іздеуі керек. Сыныптық ОББ-ны қолдану корпоративті бағдарламалық жасақтама үшін мағынасы бар шығар, өйткені ол көп өзгермейді. ОББ-дағы мінез - құлық абстрактілі сыныптарда нақты жазылған, бірақ оны сынып даналарын жасау кезінде белгілі бір дәрежеде реттеуге болады. Бұл кодты қайта пайдалануды жақсартады, бұл әзірлеушілерге көп уақытты үнемдейді. Алайда, егер сіз болашақта кодты бірнеше рет толықтырып, тіпті жобаны қайта қарауыңыз керек деп күтсеңіз, онда ОББ әзірлеушінің өнімділігіне кедергі келтіреді және код тексерілмейді, сонымен қатар қоршаған ортамен тығыз байланысты болады.