Objektitüübi laiendamine, prototüüp

Küllalt palju saab oma koodi korrastada, kui suhteliselt iseseisvalt toimivad üksused objektitüüpideks ja nende juurde kuuluvateks objektideks kokku koondada. Nii nagu tavalise järjest kirjutatud koodi kokku grupeerimisel funktsioonidesse on võimalik saada mõistlik ülevaade vähemasti kümme korda suuremast lahendusest, nii funktsioonide ja andmete koondamisel objektitüüpidesse ja objektidesse annab see omakorda vähemalt sama suure võidu programmide keerukusega hakkama saamisel. Kuni loodud uusi tüüpe on vaid mõni ning nendest loodud objektid igaüks suhteliselt erisuguste omaduste ja kasutuskohtadega, siis võivad tüübid ise rahumeeli üksteisest erinevad ja sõltumatud olla. Kui aga hakkab tekkima kohti, kus objektid peaksid käituma mõne omaduse suhtes sarnaselt, mõne omas mitte, siis võib koodi ülesehitust otstarbekamaks muuta aidata objektitüübi laiendamine.

Käskluse lisamine

Järgnevas näites luuakse kõigepealt vektori tüüp. Isendimuutujateks x ja y ning meetodiks pikkus. Hiljem lisatakse Vektorile prototüübi kaudu käsklus `tekstina()`, mis siis selle vektori andmed viisakal kujul tekstina tagastab. Vastuskihil näeb funktsiooni töö tulemust.

```
<!doctype html>
<html>
  <head>
    <title>Laiendamine</title>
    <script>
       function Vektor(x, y){
          this.x=x;
          this.y=y;
          this.pikkus=function(){
             return Math.sqrt(this.x*this.x+this.y*this.y);
          }
       }

       Vektor.prototype.tekstina=function(){
         return "("+this.x+", "+this.y+")";
       }

       function leheAlgus(){
         var autokiirus=new Vektor(3, 4);
         document.getElementById("vastus").innerHTML=
          "Tekstina: "+autokiirus.tekstina()+
          " kogukiirus "+autokiirus.pikkus();
       }
    </script>
  </head>
  <body onload="leheAlgus();">
    <div id="vastus"></div>
  </body>
</html>
```

Tekstina: (3, 4) kogukiirus 5

Massiivi viimane element

Tavapärasem on käskluse lisamine juba võõrastele tüüpidele, mida ise muuta ei saa. Näitena

lisatakse viimase elemendi küsimise käsklus Javaskripti süsteemsele klassile Array. Kui muidu on võimalik massiivist eesnimed küsida elementide arvu käsuga eesnimed.length ning viimast eesnime kujul eesnimed[eesnimed.length-1] (sest elementide lugemine algab nullist), siis funktsioonis objekti enese poole pöördumiseks on sõna this. Ehk siis viimase elemendi väärtuse annab this[this.length-1].

```
<!doctype html>
<html>
  <head>
    <title>Laiendamine</title>
    <script>
       Array.prototype.viimane=function(){
          return this[this.length-1];
       }
       var eesnimed=new Array("Juku", "Kati", "Mati");
       function leheAlgus(){
         document.getElementById("vastus").innerHTML=eesnimed.viimane();
       }
    </script>
  </head>
  <body onload="leheAlgus();">
    <div id="vastus"></div>
  </body>
</html>
```

Väljund:

Mati

Ülesandeid

- Lisa Vektorile prototüübi kaudu käsklus korruta(arv). Selle tulemusena korrutatakse vektori mõlemad koordinaadid (this.x ja this.y) vastava arvuga. Midagi ei tagastata return-käsuga. Katseta, andes vektorile algväärtused, paludes neid kahega korrutada ning siis küsides tekstina, et mis väärtused vektori sees nüüd on.
- Lisa käsklus korrutaUueks(arv). Korrutusarvutus käib samuti kummagi koordinaadi kohta. Nüüd aga ei muudeta väljakutsuva vektori andmeid, vaid luuakse käsu töö käigus uus vektor oma koordinaatidega - return new Vektor(this.x*arv, this.y*arv); Katseta.
- Uuri Javaskripti standardtüüpi String. Lisa sellele prototüübi abil teatamaks, mitmest sõnast koosneb selles stringis olev lause. Vihje: käsklus split() jagab lause sõnade massiiviks ning siis saab massiivilt pikkust küsida.

Asukoha põhjal ring

Järgnevas näites pannakse Ringi prototüübiks Asukoha eksemplar. Ehk siis siin näites tehakse Asukohast muutuja a, mille sees koordinaadid 3 ja 5. Ning käsuga

```
var a=new Asukoht(3, 5);
Ring.prototype=a;
```

määratakse, et kõik loodavad ringid saavad oma aluseks Asukoha eksemplari, millest siis ringi

loomisel koopia tehakse. Tulemusena saavad kõik ringid kasutada konkreetse asukoha koordinaate ning samuti töötab käsklus tekstina(), mis koordinaadid viisakasti sulgude vahel välja kuvab.

```
<!doctype html>
<html>
  <head>
    <title>Laiendamine</title>
    <script>
       function Asukoht(x, y){
          this.x=x;
          this.y=y;
          this.tekstina=function(){
            return "("+this.x+", "+this.y+")";
          }
       }
       var a=new Asukoht(3, 5);
       function Ring(){
          this.raadius=10;
          this.pindala=function(){
             return 3.14*this.raadius*this.raadius;
          }
       }
       Ring.prototype=a;
       var r1=new Ring();
       function leheAlgus(){
         document.getElementById("vastus").innerHTML=
           "Ringi asukoht: "+r1.tekstina()+", pindala "+r1.pindala();
       }
    </script>
  </head>
  <body onload="leheAlgus();">
    <div id="vastus"></div>
  </body>
</html>
```

Ringi asukoht: (3, 5), pindala 314

Ülesandeid

- Pane näide tööle.
- Lisa Asukohale käsklus paremale(), mis suurendab x-i väärtust ühe võrra.
- Veendu, et Ringi on võimalik ka selle käskluse abil paremale nihutada, trüki ringi asukoht enne ja pärast nihutamist.

Ülemklassi väljakutse

Eelmises näites oli ringi aluseks alati asukoht koordinaatidega 3 ja 5. Sinna saab küll käsud panna x-i ja y-i asukoha muutmiseks, aga see mõnevõrra tüütu. Hea, kui saab kohe määrata ringi sinna kus ta mõeldud on. Üheks mooduseks on ringi loomise käsu seest välja kutsuda tema aluseks oleva asukoha loomise käsk. Ehk siis luuakse kõigepealt algandmetega asukoht ja määratakse see Ringi protüübiks. Edasi Ringi loomisel antakse x ja y edasi Asukoha konstruktorile, mis hoolitseb uute koordinaatide salvestamise eest.

```
       var a=new Asukoht(0, 0);
       Ring.prototype=a;
       function Ring(x, y, r){
          Asukoht.call(this, x, y);
```

Näide tervikuna:

```
<!doctype html>
<html>
  <head>
    <title>Laiendamine</title>
    <script>
       function Asukoht(x, y){
          this.x=x;
          this.y=y;
          this.tekstina=function(){
            return "("+this.x+", "+this.y+")";
          }
       }
       var a=new Asukoht(0, 0);
       function Ring(x, y, r){
          Asukoht.call(this, x, y);
          this.raadius=r;
          this.pindala=function(){
             return 3.14*this.raadius*this.raadius;
          }
       }
       Ring.prototype=a;
       var r1=new Ring(2, 4, 7);
       function leheAlgus(){
         document.getElementById("vastus").innerHTML=
           "Ringi asukoht: "+r1.tekstina()+", pindala "+r1.pindala();
       }
    </script>
  </head>
  <body onload="leheAlgus();">
    <div id="vastus"></div>
  </body>
</html>
```

Ringi asukoht: (2, 4), pindala 153.86

Ülesandeid

- Pane näide tööle
- Lisage ringile käsklus kasPuutub(teineRing), mis väljastab, et kas üks ring puutub teisega kokku. Katseta toimimist.
- Koosta massiiv viie ringiga. Väljasta, millised ringid puutuvad esimese ringiga kokku.

Käskluse asendamine

Objektidega majandamine võimaldab uue objekti aluseks võtta ka objekti, mille mõned omadused sobivad, teised aga mitte. Siin näites luuakse kõigepealt AlusRuut, kel olemas asukoha

koordinaadid ning kiirusesamm mõlema koordinaadi suunas. Liikumisfunktsiooni käivitusega liigutakse ühe sammu jagu edasi. Joonistuskäsu tulemusena kuvatakse etteantud graafilise konteksti sihtkohale ruut, mille asukohaks alusruudu koordinaadid ning külje pikkuseks viis ühikut.

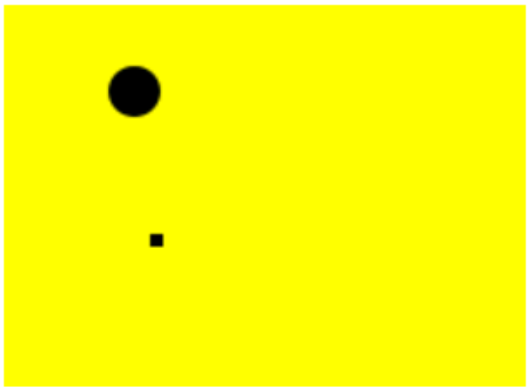
Kui nüüd soovida ringi ehk palli ekraanile joonistada ja seal liigutada, siis alusruudust sobivad kasutada asukoha koordinaadid ja liikumisearvutus, aga joonistamise tulemusena peaks midagi kandilisest ruudust ümaramat ekraanile ilmuma. Nii siin teha saabki. Kukkuva palli juures luuakse joonistuskäsklus uuesti koos ümmarguse ringiga, see asendab ruudu juurest kaasa tulnud kandilise joonistusfunktsiooni.

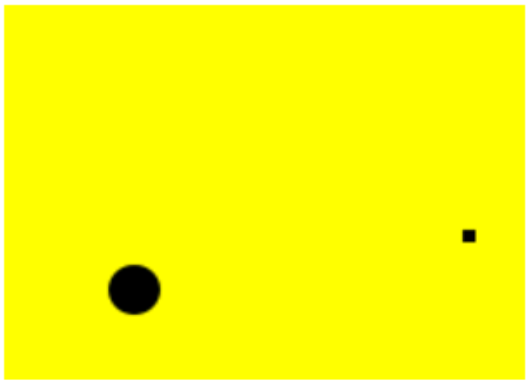
Platsi peale pannakse AlusRuut ja Pall mõlemad. Neil olemas nüüd nii joonistus- kui liikumisoskus. Pall määrab oma liikumissammuks iga kaadriga kaks ühikut allapoole, alusruudul antakse loomisel paremale liikumise andmed.

```
<!doctype html>
<html>
  <head>
    <title>Kujundid</title>
    <script>
       function AlusRuut(x, y, kiirusx, kiirusy){
          this.x=x;
          this.y=y;
          this.kx=kiirusx;
          this.ky=kiirusy;
          this.joonista=function(g){
             g.fillRect(this.x, this.y, 5, 5);
          }
          this.liigu=function(){
            this.x+=this.kx;
            this.y+=this.ky;
          }
       }
       KukkuvPall.prototype=new AlusRuut(0, 0, 0, 0);
       function KukkuvPall(x, y, r){
          AlusRuut.call(this, x, y, 0, 2)
          this.r=r;
          this.joonista=function(g){
           g.beginPath();
           g.arc(this.x, this.y, this.r,
                          0, 2*Math.PI, true);
           g.fill();
          }
       }
       function Plats(kihiId){
          window[kihiId+"_joonis"]=this;
          this.alusta=function(){
             var sisu=
               "<canvas id='"+kihiId+"_tahvel' width='200' height='150' "+
                " style='background-color: yellow' ></canvas><br/>";
             document.getElementById(kihiId).innerHTML=sisu;
             this.kujundid=new Array();
             this.tahvel=document.getElementById(kihiId+"_tahvel");
             setInterval(kihiId+"_joonis.liigu()", 1000);
          }
          this.lisaKujund=function(kujund){
            this.kujundid.push(kujund);
            this.joonista();
          }
          this.joonista=function(){
```

```
            var g=this.tahvel.getContext("2d");
            g.clearRect(0, 0, 200, 150);
            for(var i=0; i<this.kujundid.length; i++){
               this.kujundid[i].joonista(g);
            }
         }
         this.liigu=function(){
            for(var i=0; i<this.kujundid.length; i++){
               this.kujundid[i].liigu();
            }
            this.joonista();
         }
         this.alusta();
      }

      function algus(){
        kiht1_joonis=new Plats("kiht1");
        kiht1_joonis.lisaKujund(new KukkuvPall(50, 30, 10));
        kiht1_joonis.lisaKujund(new AlusRuut(50, 90, 3, 0));
      }
    </script>
  </head>
  <body onload="algus();">
     <div id="kiht1"></div>
  </body>
</html>
```





Ülesandeid

- Pane näide tööle
- Lisa mitu ruutu erisuguste kiirustega
- Kukkuva palli juures määra kukkumiskiirus juhuslikult vahemikus 1-5 ühikut kaadri kohta
- Lisa katsetamiseks mitu palli.

Klassi prototüübid andmehalduses

Järgnevalt pikem näide, kus eri tegevused eri objektide vahel jaotatud ning võimaldavad kummalgi tüübil selle jaoks mõeldud tegevusele keskenduda ning sellega koodi paremini loetavana hoida. Alustuseks luuakse AndmeHalduse tüüp, kus talletatakse teksti ja arvu paarid ning kust saab hiljem arvud eraldi funktsiooniga välja küsida.

```
<!doctype html>
<html>
  <head>
    <title>Kujundid</title>
    <script>
       function AndmeHaldus(){
          this.andmed=new Array();
          this.lisaKirje=function(tekst, arv){
             this.andmed.push({"t": tekst, "a": arv});
          }
          this.kysiArvud=function(){
             var arvud=new Array();
             for(var i=0; i<this.andmed.length; i++){
                arvud.push(this.andmed[i].a);
             }
             return arvud;
          }
       }

       function algus(){
          var ah=new AndmeHaldus();
          ah.lisaKirje("Juku", 175);
          ah.lisaKirje("Kati", 165);
          document.getElementById("kiht1").innerHTML=ah.kysiArvud();
       }
    </script>
  </head>
  <body onload="algus();">
     <div id="kiht1"></div>
  </body>
</html>
```

Näite väljundiks siis kaks arvu:

175,165

Tabelina näitav laiendus

Andmete hoidjale saab vajadust mööda mitmesuguseid andmete kuvajaid peale ehitada. AndmeTabeli objektis jäetakse meelde teksti ja arvu komplektid. Eraldi käsuga saab välja küsida ka ainult arvud. Olemasolevale objektile saab peale ehitada teise.

```
AndmeTabel.prototype=new AndmeHaldus();
```

ütleb, et igale uuele AndmeTabelile võetakse aluseks AndmeHalduse eksemplar. Ehk siis AndmeTabeli objekt oskab samuti enese sisse teksti ja arvu komplekte koguda ja sealt arve eraldada. Lisaks on tal juures oskus tulemuste kuvamiseks tabelina.

```
<!doctype html>
<html>
  <head>
    <title>Kujundid</title>
    <script>
       function AndmeHaldus(){
          this.andmed=new Array();
          this.lisaKirje=function(tekst, arv){
             this.andmed.push({"t": tekst, "a": arv});
          }
```

```
            this.kysiArvud=function(){
               var arvud=new Array();
               for(var i=0; i<this.andmed.length; i++){
                  arvud.push(this.andmed[i].a);
               }
               return arvud;
            }
         }



         AndmeTabel.prototype=new AndmeHaldus();
         function AndmeTabel(){
            this.HTMLTabelina=function(){
               var t="<table>";
               for(var i=0; i<this.andmed.length; i++){
                 t+="<tr><td>"+this.andmed[i].t+"</td><td>"+
                                this.andmed[i].a+"</td></tr>\n";
               }
               t+="</table>";
               return t;
            }
         }

         function algus(){
            var at=new AndmeTabel();
            at.lisaKirje("Juku", 175);
            at.lisaKirje("Kati", 165);
            document.getElementById("kiht1").innerHTML=at.HTMLTabelina();
         }
      </script>
   </head>
   <body onload="algus();">
      <div id="kiht1"></div>
   </body>
</html>
```

Tabelit võib lehel vaadata

Juku 175

Kati 165

Ülesandeid

- Pane näide tööle
- Lisa AndmeTabeli juurde käsk arvude loetelu näitamiseks (unordered list)

Joonisena näitav laiendus

Samale AndmeHalduse tüübile võimalik peale ehitada teine lahendus, sedakorda joonistusoskusega. Siin kasutatakse arvudena välja küsimise oskust. Joonistustahvel lisatakse funktsioonis etteantud nimega kihile sedakorda Javaskripti DOM-funktsioonide abil.

```
            var tahvel=document.createElement("canvas");
```

```
            tahvel.setAttribute("width",  "300");
            tahvel.setAttribute("height", "200");
            tahvel.style.backgroundColor="yellow";
            document.getElementById(kihinimi).appendChild(tahvel);
```

Üheks mooduseks elementide veebilehele lisamisel on lehe elementide omadus innerHTML, kuhu kirjutatud väärtused teisendatakse brauseri mälus seal olevateks objektideks. Samas on aga võimalik ka objektipuud mööda liikuda ja seal andmeid elemente luua ja ümber paigutada, mis mõnel juhul on mugavam või kiirem. Sõltumata loomismoodusest saab tahvlit ikka ühtemoodi kasutada.

```
<!doctype html>
<html>
  <head>
    <title>Kujundid</title>
    <script>
       function AndmeHaldus(){
          this.andmed=new Array();
          this.lisaKirje=function(tekst, arv){
             this.andmed.push({"t": tekst, "a": arv});
          }
          this.kysiArvud=function(){
             var arvud=new Array();
             for(var i=0; i<this.andmed.length; i++){
                arvud.push(this.andmed[i].a);
             }
             return arvud;
          }
       }

       AndmeJoonis.prototype=new AndmeHaldus();
       function AndmeJoonis(){
          this.arvudJoonisena=function(kihinimi){
             var tahvel=document.createElement("canvas");
             tahvel.setAttribute("width",  "300");
             tahvel.setAttribute("height", "200");
             tahvel.style.backgroundColor="yellow";
             document.getElementById(kihinimi).appendChild(tahvel);
             var g=tahvel.getContext("2d");
             var arvud=this.kysiArvud();
             for(var i=0; i<arvud.length; i++){
                g.fillRect(10, 10+i*20, arvud[i], 10);
             }
          }
       }


       function algus(){
          var aj=new AndmeJoonis();
          aj.lisaKirje("Juku", 175);
          aj.lisaKirje("Kati", 165);
          aj.arvudJoonisena("kiht1");
//          document.getElementById("kiht1").innerHTML=at.HTMLTabelina();
       }
    </script>
  </head>
  <body onload="algus();">
     <div id="kiht1"></div>
  </body>
</html>
```

Tulemusena võibki imetleda andmete graafilist väljundit.

Ülesandeid

- Pane näited tööle
- Kuva AndmeJoonise tulpade juurde ka nimetused ja arvud