



휴먼컴퓨터 인터페이스

과제 #1. 기본 위젯 구현

2020.04.29

컴퓨터 소프트웨어 학과

2016726009 | 임현우

01

요구조건과 제약조건 만족도 Summary

요구조건 1

WebUI

Widget

Text

Image

PushButton

TextField

코드 입력

○

○

○

○

○

○

빠진 부분
찾기

- ▷ initWidget
- ▷ layoutWidgets
- ▷ handleMouseDown
- ▷ handleMouseMove
- ▷ handleMouseUp
- ▷ findWidgetsOn

- ▷ Widget
- ▷ translate

- ▷ Text
- ▷ initVisualItems

- ▷ initVisualItems

- ▷ handleMouseDown
- ▷ handleMouseUp
- ▷ handleMouseEnter
- ▷ handleMouseExit

- ▷ initVisualItems

01

요구조건과 제약조건 만족도 Summary

요구조건2

Switch

함수 이름	Switch	initVisualItems	handleMouseDown
기능	Switch 객체의 속성 초기화	Path 객체와 circle 객체를 과제에 주어진 조건에 맞춰 생성하여 visual_items배열에 추가	Switch객체 위에서 mouseDownEvent 발생 시 switch의 is_on 속성을 바꾸어 주며 circle 객체를 이동시키고 배경색에 변화를 주도록 함.

01

요구조건과 제약조건 만족도 Summary

제약 조건

1	클라이언트 측 스크립트만 사용	○
2	외부 라이브러리 제한적 허용(jQuery와 fabric.js만 허용)	○
3	외부 리소스 사용X	○
4	구글 크롬 웹 브라우저 호환 필수	○

initWidgets

```
WebUI.initWidgets = function() {  
    WebUI.title = new WebUI.Text("introduction to HCI");  
  
    WebUI.img_html = new WebUI.Image("resources/HTML5.png", {width: 100, height: 80});  
    WebUI.img_css = new WebUI.Image("resources/CSS3.png", {width:100, height:80});  
    WebUI.img_js = new WebUI.Image("resources/JS.png", {width: 100, height: 80});  
  
    WebUI.text_id = new WebUI.Text("ID");  
    WebUI.text_pwd = new WebUI.Text("Password");  
  
    WebUI.edit_id = new WebUI.TextField("", {width: 200, height:50});  
    WebUI.edit_pwd = new WebUI.TextField("", {width:200, height:50});  
  
    WebUI.btn_ok = new WebUI.PushButton("OK", {width:100, height:50});  
    WebUI.btn_cancel = new WebUI.PushButton("Cancel", {width:100, height:50});  
  
    WebUI.text_blah = new WebUI.Text("I want to get A+!");  
    WebUI.switch = new WebUI.Switch(false, {width: 100, height:50});  
}
```

로그인 화면에 필요한 객체들을 생성하는 단계이다. Text, Image, TextField, PushButton, Switch 객체를 생성하고 크기 값을 초기화 하였다.

02

각 객체의 구현 방법 설명 - WebUI

layoutWidgets

```
WebUI.layoutWidgets = function() {  
    WebUI.title.moveTo({left: 100, top: 10});  
  
    WebUI.img_html.moveTo({left: 50, top: 50});  
    WebUI.img_css.moveTo({left: 160, top: 50});  
    WebUI.img_js.moveTo({left: 270, top: 50});  
  
    WebUI.text_id.moveTo({left: 50, top: 160});  
    WebUI.text_pwd.moveTo({left: 50, top: 220});  
  
    WebUI.edit_id.moveTo({left: 150, top: 140});  
    WebUI.edit_pwd.moveTo({left: 150, top: 200});  
  
    WebUI.text_blah.moveTo({left: 50, top: 300});  
    WebUI.switch.moveTo({left: 250, top: 280});  
  
    WebUI.btn_ok.moveTo({left: 50, top: 350});  
    WebUI.btn_cancel.moveTo({left: 160, top: 350});  
}
```

initWidgets 함수에서 생성한 객체들을 지정된 좌표로 이동시키는 함수이다.

handleMouseDown

```

WebUI.handleMouseDown = function(window_p) {
  let is_handled=false;
  if(WebUI.isInCanvas(window_p)){
    let canvas_p = WebUI.transformToCanvasCoords(window_p);

    WebUI.is_mouse_dragging = true;
    WebUI.mouse_drag_start = canvas_p;
    WebUI.mouse_drag_prev = canvas_p;

    let widget = WebUI.findWidgetOn(canvas_p);
    if(widget){
      WebUI.focused_widget = widget;

      if(widget.is_draggable){
        WebUI.dragged_widget = widget;
      }
      else{
        WebUI.dragged_widget = null;
      }
      is_handled = widget.handleMouseDown(canvas_p);
    }
    else{
      WebUI.focused_widget = null;
      WebUI.dragged_widget = null;
    }
  }
  else{
    WebUI.is_mouse_dragging = false;
    WebUI.mouse_drag_start = {x:0, y:0};
    WebUI.mouse_drag_prev = {x:0, y:0};

    WebUI.focused_widget = null;
    WebUI.dragged_widget = null;
  }

  if(is_handled){
    WebUI.canvas.requestRenderAll();
  }
}

```

mouseDown event를 처리하는 함수의 기초적인 단계이다.

isInCanvas 함수를 통해 canvas 내부에서 발생한 이벤트인지 확인하고 transformToCanvasCoords 함수를 이용해 window 좌표를 canvas 좌표로 변환한다. 변환된 canvas 좌표를 이용해 클릭 이벤트가 발생한 위젯을 찾아 해당 위젯의 handler로 이벤트를 위임한다.

mouseDown event가 발생하면, 객체를 드래그할 일이 발생할 수 있기 때문에, 해당 기능도 기초적인 단계가 구현되어 있다.

handleMouseMove

```
WebUI.handleMouseMove = function(window_p) {
  let is_handled = false;
  let canvas_p = WebUI.transformToCanvasCoords(window_p);

  let widget = WebUI.findWidgetOn(canvas_p);
  if(widget != WebUI.hovered_widget){
    if(WebUI.hovered_widget != null){
      is_handled = is_handled || WebUI.hovered_widget.handleMouseExit(canvas_p);
    }
    if(widget != null){
      is_handled = is_handled || widget.handleMouseEnter(canvas_p);
    }
    WebUI.hovered_widget = widget;
  }
  else{
    if(widget){
      is_handled = widget.handleMouseMove(canvas_p);
    }
  }
  if(WebUI.is_mouse_dragging){
    if(WebUI.dragged_widget != null){
      let tx = canvas_p.x - WebUI.mouse_drag_prev.x;
      let ty = canvas_p.y - WebUI.mouse_drag_prev.y;
      WebUI.dragged_widget.translate({x: tx, y: ty});

      is_handled = true;
    }
    WebUI.mouse_drag_prev = canvas_p;
  }
  if(is_handled){
    WebUI.canvas.requestRenderAll();
  }
}
```

Mouse로 객체를 클릭한 뒤 드래그 할 때 발생하는 이벤트를 다루는 핸들러이다. 드래킹 되고 있는 위젯의 위치를 이동시키기 위해 translate 함수의 호출 부분이 있다.

handleMouseUp

```
WebUI.handleMouseUp = function(window_p) {  
  let is_handled = false;  
  let canvas_p = WebUI.transformToCanvasCoords(window_p);  
  
  let widget = WebUI.findWidgetOn(canvas_p);  
  if(widget){  
    is_handled = widget.handleMouseUp(canvas_p);  
  }  
  
  if(WebUI.is_mouse_dragging){  
    WebUI.is_mouse_dragging = false;  
    WebUI.mouse_drag_start = {x:0, y:0};  
    WebUI.mouse_drag_prev = {x:0, y:0};  
  
    WebUI.dragged_widget = null;  
  
    is_handled = true;  
  }  
  
  if(is_handled){  
    WebUI.canvas.requestRenderAll();  
  }  
}
```

mouseDown 이후 마우스를 땄 때 발생하는 이벤트를 다루는 기초적인 이벤트 핸들러이다. mouseDown, mouseMove 이벤트를 다루기 위해 초기화 했던 변수들을 다시 디폴트 값으로 변경해 주는 단계가 있으며 mouseDown, mouseMove, mouseUp 이벤트는 모두 위젯의 상태를 변경시킬 수 있으므로 위 핸들러 들은 마지막에 전부 requestRenderAll 함수를 호출한다.

02

각 객체의 구현 방법 설명 - WebUI

findWidgetOn

```
//canvas_p(클릭지점)에 있는 widget 반환
WebUI.findWidgetOn = function(canvas_p) {
  let x = canvas_p.x;
  let y = canvas_p.y;

  for(let i=0; i<this.widgets.length; i++){
    let widget = this.widgets[i];

    if(x >= widget.position.left &&
       x <= widget.position.left + widget.size.width &&
       y >= widget.position.top &&
       y <= widget.position.top + widget.size.height){
      return widget;
    }
  }
  return null;
}
```

findWidgetOn 함수는 window 좌표가 canvas 위의 좌표로 변환된 값인 canvas_p를 인자로 받아 canvas_p가 있는 위치에 존재하는 widget을 찾아 반환한다.

02

각 객체의 구현 방법 설명 - Widget

Widget

```
WebUI.Widget = function() {  
  this.type = WebUI.WidgetTypes.UNDEFINED;  
  
  this.parent = null;  
  this.children = [];  
  
  this.position = {left: 0, top: 0};  
  this.size = {width: 0, height: 0};  
  
  this.is_draggable = false;  
  this.is_movable = true;  
  
  this.visual_items = [];  
  this.is_resource_ready = false;  
  
  WebUI.widgets.push(this);  
}
```

Widget 객체가 기본적으로 가져야할 속성들의 초기값을 입력해주는 함수이다. 초기값을 입력해 주고 나면 모든 Widget 객체들이 저장되는 배열인 widgets에 해당 widget을 push해 준다.

02

각 객체의 구현 방법 설명 - Widget

translate

```
WebUI.Widget.prototype.translate = function(v) {  
  if(!this.is_movable) return;  
  
  this.position.left += v.x;  
  this.position.top += v.y;  
  this.visual_items.forEach(item => {  
    item.left += v.x;  
    item.top += v.y;  
  });  
  this.children.forEach(child_widget => {  
    child_widget.translate(v);  
  });  
}
```

Widget 객체를 이동시키기 위한 함수로써 이동 시키고자 하는 값인 v를 인자로 받아 widget 객체의 위치 속성을 변경시키고 해당 widget이 가지고 있는 자식 객체들도 같은 크기만큼 이동 시킨다.

02

각 객체의 구현 방법 설명 - Text

Text

```
//Text widget
WebUI.Text = function(label) {
  WebUI.Widget.call(this);

  this.type = WebUI.WidgetTypes.TEXT;
  this.label = label;

  this.font_family = 'System';
  this.font_size = 20;
  this.font_weight = 'bold';
  this.text_align = 'left';
  this.text_color = 'black';
}
```

Text 객체가 갖는 속성들의 초기값을 입력해주는 함수이다. WebUI.Widget.call(this); 문을 통해서 Widget 자체의 생성자 또한 호출하며 Widget 생성자에게 자기자신을 전달한다.

initVisualItems

```
WebUI.Text.prototype.initVisualItems = function(){
  let text = new fabric.Text(this.label, {
    left: this.position.left,
    top: this.position.top,
    selectable: false,
    fontFamily: this.font_family,
    fontSize: this.font_size,
    fontWeight: this.font_weight,
    textAlign: this.text_align,
    stroke: this.text_color,
    fill: this.text_color
  });

  let bound = text.getBoundingBox();
  this.position.left = bound.left;
  this.position.top = bound.top;
  this.size.width = bound.width;
  this.size.height = bound.height;

  this.visual_items.push(text);
  this.is_resource_ready = true;
}
```

Text객체가 갖는 fabric 객체인 fabric.Text 객체를 생성하고, visual_items 배열에 push 한다. 또한 bound 변수에 text객체의 위치와 크기 값을 저장한다.

initVisualItems

```
WebUI.Image.prototype.initVisualItems = function() {  
  let widget = this;  
  
  fabric.Image.fromURL(this.path, function(img){  
    if(widget.desired_size != undefined){  
      img.scaleToWidth(widget.desired_size.width);  
      img.scaleToHeight(widget.desired_size.height);  
      widget.size = widget.desired_size;  
    }  
    else{  
      widget.size = {width: img.width, height: img.height};  
    }  
    img.set({left:widget.position.left,  
            top: widget.position.top,  
            selectable: false});  
    widget.visual_items.push(img);  
    widget.is_resource_ready = true;  
  });  
}
```

Fabric.Image 객체를 생성하고 그 크기를 결정하는 함수이다. Image는 Url을 전달받아 불러오게 되며, 불러온 이미지를 visual_items배열에 push 한다.

02

각 객체의 구현 방법 설명 – Push Button

handleMouseDown

```
WebUI.PushButton.prototype.handleMouseDown = function() {  
  if(!this.is_pushed){  
    this.translate({x:0, y:5});  
    this.is_pushed = true;  
  
    if(this.onPushed != undefined){  
      this.onPushed.call(this);  
    }  
    return true;  
  }  
  else{  
    return false;  
  }  
}
```

Push Button 객체에서 Mouse Down Event가 발생할 때 호출되는 이벤트 핸들러이다. 버튼이 눌렸음을 시각적으로 보여주기 위해서 버튼이 5px 만큼 아래로 translate 되는 기능이 구현되어 있다.

02

각 객체의 구현 방법 설명 – Push Button

handleMouseUp

```
WebUI.PushButton.prototype.handleMouseUp = function() {  
  if(this.is_pushed){  
    this.translate({x:0, y: -5});  
    this.is_pushed = false;  
  
    return true;  
  }  
  else{  
    return false;  
  }  
}
```

Push Button 객체에서 Mouse Up Event가 발생할 때 호출되는 이벤트 핸들러이다. 버튼이 눌렀다가 떴음을 시각적으로 보여주기 위해 버튼이 5px 만큼 위로 translate 되는 기능이 구현되어 있다.

02

각 객체의 구현 방법 설명 – Push Button

handleMouseEnter

```
WebUI.PushButton.prototype.handleMouseEnter = function() {  
    this.visual_items[0].set('strokeWidth', 3);  
    return true;  
}
```

Push Button 객체가 차지하는 공간에 마우스 포인터가 들어왔을 때 호출되는 이벤트 핸들러이다. 마우스 포인터가 들어온 것을 시각적으로 보여주기 위해 Push Button의 Stroke Width 속성을 조절하는 기능이 구현되어 있다.

02

각 객체의 구현 방법 설명 – Push Button

handleMouseExit

```
WebUI.PushButton.prototype.handleMouseExit = function() {  
    this.visual_items[0].set('strokeWidth', 1);  
  
    if(this.is_pushed){  
        this.translate({x:0, y:-5});  
        this.is_pushed = false;  
    }  
    return true;  
}
```

Push Button 객체가 차지하는 공간에 마우스 포인터가 들어왔다가 나갈 때 호출되는 이벤트 핸들러이다. 마우스 포인터가 나가는 것을 시각적으로 보여주기 위해 Push Button의 Stroke Width 속성을 조절하는 기능이 구현되어 있다.

02

각 객체의 구현 방법 설명 – Text Field

initVisualItems

```
WebUI.TextField.prototype.initVisualItems = function() {
  let boundary = new fabric.Rect({
    left: this.position.left,
    top: this.position.top,
    width: this.desired_size.width,
    height: this.desired_size.height,
    fill: this.fill_color,
    stroke: this.stroke_color,
    strokeWidth: this.stroke_width,
    selectable: false
  });
  let textbox = new fabric.Textbox(this.label, {
    left: this.position.left + this.margin,
    fontFamily: this.font_family,
    fontSize: this.font_size,
    fontWeight: this.font_weight,
    textAlign: this.text_align,
    stroke: this.text_color,
    fill: this.text_color,
    selectable: false
  });
  let bound = textbox.getBoundingBox();
  textbox.top = this.position.top + (this.desired_size.height - bound.height)/2;

  this.size = this.desired_size;

  this.visual_items.push(boundary);
  this.visual_items.push(textbox);

  this.is_resource_ready = true;
}
```

Text Field 객체가 가져야 할 fabric.Textbox 객체를 생성하고, 그 객체의 크기와 위치를 조정해주는 함수이다. TextField 내부에 글자가 입력될 공간인 fabric.Textbox와 빈 공간인 fabric.Rect의 위치를 지정한다. 이후에 두 객체를 visual_items배열에 추가하는 기능이 구현되어 있다.

02

각 객체의 구현 방법 설명 - Switch

Switch

```
//Switch 객체 생성////////////////////////////////////
WebUI.Switch = function(is_on, desired_size) {
  WebUI.Widget.call(this);
  this.type = WebUI.WidgetTypes.SWITCH;

  this.is_on = is_on;
  this.desired_size = desired_size;
  this.radius = desired_size.width / 4;

  this.fill_color = 'rgb(142,142,147)';
  this.stroke_color = 'rgb(142,142,147)';
}
```

Switch 객체를 생성할 때 호출되는 함수이다. 스위치가 켜져 있는지 아닌지에 대한 값인 is_on 속성을 포함하고 있으며, circle 객체를 갖는 widget 이므로 radius 속성도 초기화 한다.

02

각 객체의 구현 방법 설명 – Switch

initVisualItems

```
WebUI.Switch.prototype.initVisualItems = function() {
  let itself = this;
  let radius = this.desired_size.width / 4;
  let divTimes = 30;0.
  let divAngle = Math.PI/divTimes;
  //String(this.position.left+radius) + ',' + String(this.position.top)
  let path = new fabric.Path('M' + radius + ',0' +
  function(){
    let arcPath = "";
    for(let i=0; i<divTimes; i++){
      arcPath += 'L' + String(radius-radius*Math.sin(i*divAngle)) + ',' + String(radius-radius*Math.cos(i*divAngle));
    }
    arcPath += 'L' + String(3*radius) + ',' + String(2*radius);
    for(let i=0; i<divTimes; i++){
      arcPath += 'L' + String(3*radius + radius*Math.sin(i*divAngle)) + ',' + String(radius+radius*Math.cos(i*divAngle));
    }
    arcPath += 'L' + String(radius) + ',0';
    return arcPath;
  }());

  let circle = new fabric.Circle({
    left: this.position.left+ radius*0.1,
    top: this.position.top + radius*0.1,
    radius: radius * 0.9,
    fill: 'rgb(255,255,255)',
    stroke: 'rgb(142,142,147)',
    selectable: false
  });
  this.size = this.desired_size;
  path.set({ fill: 'rgb(142,142,147)', stroke: 'rgb(142,142,147)' });

  this.visual_items.push(path);
  this.visual_items.push(circle);
  this.is_resource_ready = true;
}
```

Switch 객체가 fabric.path와 fabric.circle 객체를 포함하도록 구현하였다.

Fabric.Path 객체는 곡선을 그려주기 위해서 삼각함수를 사용 하였고 익명함수가 곡선의 경로를 반환할 수 있도록 구현하였다.




handleMouseDown

```
WebUI.Switch.prototype.handleMouseDown = function() {  
  if(this.is_on == true){  
    this.visual_items[1].animate('left', '--' +(2*this.radius), {  
      onChange: WebUI.canvas.renderAll.bind(WebUI.canvas),  
      duration: 100,  
    });  
    this.visual_items[0].set('stroke', 'rgb(142,142,147)');  
    this.visual_items[0].set('fill', 'rgb(142,142,147)');  
    this.visual_items[1].set('stroke', 'rgb(142,142,147)');  
    this.is_on = false;  
  }  
  else{  
    this.visual_items[1].animate('left', '+=' +(2*this.radius), {  
      onChange: WebUI.canvas.renderAll.bind(WebUI.canvas),  
      duration: 100,  
    });  
    this.visual_items[0].set('stroke', 'rgb(48,209,88)');  
    this.visual_items[0].set('fill', 'rgb(48,209,88)');  
    this.visual_items[1].set('stroke', 'rgb(48,209,88)');  
    this.is_on = true;  
  }  
  return true;  
}
```

Switch 객체 위에서 Mouse Down Event
가 발생할 때 호출되는 함수이다. Switch
객체가 토글 형식으로 클릭 할 때마다 꺼
지고 켜져야 하므로 is_on 속성을
true/false로 바꿈과 동시에 fabric.circle
객체와 fabric.path 객체의 색상도 달라지
게 구현 하였다.

introduction to HCI

HTML **CSS** **JS**



ID

Password



I want to get A+! ☐

OK **Cancel**

초기 화면
구현한 모든 객체가
누락된 것 없이 모두 화면에 표시 되었음

introduction to HCI

HTML **CSS** **JS**

ID

Password

I want to get A+! ☐

OK **Cancel**




텍스트 입력
및
문자열이 텍스트 박스를 넘어가지 않는 모습

03

테스트 결과 제시 및 분석

introduction to HCI

HTML **CSS** **JS**

ID

Password

I want to get A+! ☒

Switch 클릭 시 활성화 된 시각적 효과

introduction to HCI

HTML CSS JS

ID ImHyunWoo

Password 2016726009

I want to get A+! ☐

OK Cancel

마우스를 Push Button 위에 올려 놓았을 때
테두리가 두껍게 변하는 시각적 효과

introduction to HCI

HTML CSS JS

ID ImHyunWoo

Password humancomputer

I want to get A+! ☒

OK Cancel

논의

- ① 과제에서 요구한 모든 객체의 생성과 객체의 기능을 완성하였다. 특히 Switch 내부의 fabric.Path 객체를 생성함에 있어서 삼각 함수를 사용한 경로 출력이 가장 성공적인 기능 구현이라고 생각한다.
- ② 자체적인 평가를 하자면 **과제에서 요구하는 모든 요건을 충족**하였기 때문에 양호한 수준이라고 보여진다.
- ③ **향후 개선점**
 - 자식 객체에 대한 연습을 해보기 위하여 초기에는 Switch 객체 안에 fabric.Circle과 fabric.Path 객체를 자식 객체로써 구현하려고 하였으나 실패하였다. 향후에는 자식 객체에 대한 이해를 높여 다시 도전할 수 있을 것이다.



감사합니다