Lektion 8: Bob Ross

HTMLCanvasElement: eine rechteckige Fläche auf einer Browserseite, deren Pixel per Code beliebig gefärbt werden können => Formen und Farben lassen sich dynamisch generieren
Um das HTMLCanvasElement zu manipulieren, wird die Programmierschnittstelle (API := Application Programming Interfaces) CanvasRenderingContext verwendet. Diese gibt es sowohlt für 2D- als auch 3D-Grafikdarstellung
Canvas-Element initial immer mit #FFFFFF (weißen) Pixeln besetzt

Farbverläufe

Natürlich sind auch Farbverläufe möglich. Dafür verwendet man einen CanvasGradient:

Let gradient: CanvasGradient = crc2.createLinearGradient(0, 0, 0, 100);

Der Farbverlauf zieht sich hier stumpf senkrecht von oben 100 Pixel nach unten. Damit der gradient auch weiß, von welcher über welche zu welcher Farbe der Verlauf gehen soll, werden ColorStops hinzugefügt:

```
gradient.addColorStop(0, „pink“);
gradient.addColorStop(.5, „yellow“);
gradient.addColorStop(1, „blue“);
```

Die Zahlenwerte (quasi) Prozentangaben der Strecke des gradient und sagen, bei wie viel Prozent der Strecke welche Farbe erreicht sein soll. Dabei ist 0 immer der Start, 1 das Ziel und sämtliche Kommazahlen dazwischen der entsprechende Streckenteil dazwischen. In diesem Fall Ist unser Verlauf am Anfang pink, in der Mitte gelb und am Ende blau (ist die Panflagge, wann gucken wir wieder The Owl House?). gradient kann jetzt wie eine Farbe dem crc2.fillStyle zugewiesen werden.

Pattern

Man kann auch Füllmuster einsetzen. Dafür kann man Bitmaps, SVG-Bilder oder auch einfach Grafiken des Canvas verwenden. Letztere werden dem fillStyle wie folgt zugewiesen:

```
//wir machen erst ein Muster
let pattern: CanvasRenderingContext = document.createElement(‚canvas‘).getContext(‚2d‘)
pattern.width = whatever
pattern.height = whatever

/** es wird was gezeichnet*/

crc2.fillStyle = crc2.createPatter(pattern.canvas, repeat)
```

Know your canvas!

Befehl	**Was macht das**
fillStyle = „#00FF00“	Weist eine Füllfarbe zu
fillRect(x, y, width, height)	Erstellt ein Rechteck bei (x | y), das width weit und height hoch ist und mit der Füllfarbe gefüllt ist
clearRect(x, y, width, height)	Erstellt ein Rechteck bei (x | y), das width weit und height hoch ist und die Pixel weiß füllt
strokeRect(x, y, width, height)	Erstellt ein Rechteck bei (x | y), das width weit und height hoch ist und einen gefärbten Rand hat
stroke()	Färbt die Pixel eines Pfades
fill()	Färbt die Pixel innerhalb eines Pfades
arc(x, y, r, sAngle, eAngle)	Erstellt eine Kurve ausgehend vom Punkt (x | y) mit dem Radius r einem Start- und Endwinkel des Einheitskreises (ein Kreis geht vom Winkel 0 zum Winkel 2 PI)
beginPath()	Öffnet einen Pfad
closePath()	Schließt einen Pfad
ellipse()	Ähnlich wie arc()
moveTo(x, y)	Setzt springt innerhalb des Pfades zu diesem Punkt
lineTo(x, y)	Zieht eine Linie zu diesem Punkt
translate()	Verschiebt anhand der Gesamtmatrix den Ursprung (O = (0 | 0)) nach (x | y).
rotate(angle)	Rotiert das Bild um 0 – 2 PI
scale(scalewidth, scaleheight)	Skaliert das Bild auf die angegebenen Maße
resetTransform()	Bring die Gesamtmatrix auf den Urzustand
save()	Speichert den aktuellen Zustand der Gesamtmatrix. Kann auch mehrmals getan werden
restore()	Stellt den gespeicherten Zustand der Gesamtmatrix wieder her. Sollten mehrere Zustände gespeichert sein, werden diese in umgekehrter Reihenfolge restauriert
getTransform()	Speichert den aktuellen Zustand der Gesamtmatrix zurück und weißt ihn einer Variable vom Typ DOMMatrix zu
setTransform(_matrix: DOMMatrix)	Stellt den als Parameter angegebenen Zustand der Gesamtmatrix wieder her
bezierCurveTo(cp1x, cp1y, cp1x, cp2y, x, y)	Zieht eine Kurve nach (x, y), die anhand der Bézier-Kontrollpunkte gekrümmt wird
quadraticCurveTo(cpx, cpy, x, y)	Zieht eine Parabel nach (x, y), deren Hoch-/Tiefpunkt bei (cpx | cpy) liegt

Lektion 9: Gott ist ein DJ

Es folgen unsere neuen Freunde: die Klassen! Mithilfe von Klassen tauchen wir in die Objektorientierte Programmierung (OOP(s I did it again)), bzw in unserem Fall in das OO Design ein. Anstatt also stur drauf los ein Schiff, ein Flugzeug, drei Pandas und 20 Chapatis zu coden und unser Skript unnötig unübersichtlich und lang zu machen, Erstellen wir eine jeweilige Klasse und instanziieren ein Objekt aus dieser. HOW DOE?!? Let me tell you! In der Folgenden Erklärung baue ich eine Flugzeugklasse.

Abstraktion

Zunächst abstrahieren wir ein imaginäres Objekt unserer Klasse anhand dieser fünf Fragen:
Was hat es?
Was kann es?
Was weiß es?
Wer hält es?
Was ist es?

Das können wir dann modellieren und in einem Klassendiagramm darstellen

Airplane
type: string tankLevel: number tankLimit: number airport: string
constructor(_type: string, tankLimit: number, _airport: string) refuel(): void fly(_destination: string): void

Anhand unseres Modells können wir uns an die Implementation wagen. In diesem Fall sieht diese so aus:

```
class Airplane {

        type: string;
        tankLevel: number;
        tankLimit: number;
        airport: string;
        constructor(_type: string, tankLimit: number, _airport: string) {

                kümmern wir uns nachher drum

        }
        refuel(): void {this.tankLevel = tankLimit;
        }
        fly(_destination: string): void {

                this.airport = _destination;
                this.tankLevel -= 100;
                }

}
```

Da jetzt die Klasse erstellt ist, muss das Hauptprogramm nicht mehr wissen, **wie** man ein Flugzeug fliegt oder es tankt sondern nur noch, **dass** man es fliegen und tanken kann. Das übernimmt das Flugzeug. Jetzt geht es nur noch darum ein Flugzeug zu instanziieren:

```
let boeing747: Airplane = new Airplane("boeing747", 20000, "Fraport")
```

Dieses Flugzeug kann jetzt die Methoden refuel() und fly() verwenden:

```
boeing747.fly(„Amsterdam“);
boeing747.refuel();
```

Die Methoden haben wir im oberen Beispiel mit this. gespickt. Das liegt daran, dass die Methoden sich dann immer auf das gezielte Objekt beziehen. Die Methode für unseren Flug in die Niederlande kann man sich ausgeschrieben jetzt also so vorstellen:

```
boeing747.fly(„Amsterdam“) {

    boeing747.destination = "Amsterdam"
    boeing747.tankLevel -= 100

}
```

„Eine Klasse beschreibt eine Struktur, die ein Objekt derselben aufweisen soll und die Methoden, die mit einem solchen verknüpft werden.“ Ein solches Objekt wird nach einem der Klasse entsprechenden Bauplan konstruiert. Bühne frei für den

Constructor

Der Constructor beschreibt, was bei der Instanziierung eines Objekts einer Klasse getan werden muss, um das Objekt korrekt als solches zu konstruieren. Dabei können z.B. Werte über Parameter an das Objekt gegeben und entsprechende Methoden ausgeführt werden. In unserem Fall erschaffen wir ein Flugzeug, das mit einer Modellnummer, einer Tankkapazität und einem Flughafen, in dem es steht, ausgestattet ist. Danach soll es volltanken. In unserer Klasse sähe der Constructor dann so aus:

```
constructor(_type: string, _tankLimit: number, _airport: string) {

    this.type = _type;
    this.tankLimit = _tankLimit;
    this.airport = _airport;

    this.refuel();

}
```

Ein Constructor kann natürlich auch nichts tun und ohne Parameter gemacht werden. In diesem Fall muss man ihn nicht extra schreiben, da TypeScript immer automatisch einen Constructor bereitstellt, der aber nichts macht.

Nachdem wir nun den Gott-Teil des Kapitels durchgenommen, muss noch der DJ-Teil geklärt werden. Dafür beschäfigt uns das nächste Thema:

Zeitsignale

Ein Animation per Schleife ist nicht immer machbar, da der Browser dann nur noch mit der Schleife und nicht mit dem Bild beschäftigt ist. Lieber soll er das Bild zeigen, dann ein bisschen warten und dann das nächste Bild zeigen, usw. Uns stehen folgende Zeitsignale zur Verfügung:

window.setTimeout – Einmal machen, nie mehr schaffen
window.setTimeout(handler, time, args…) bewirkt, dass die handler-Funktion nach der in Millisekunden angegebenen Zeit ein mal ausgeführt wird. Weitere Parameter (args…) können der Funktion dabei übergeben werden.

window.setIntervall – Loop Button gefunden
window.setIntervall(handler, time, args…) bewirkt, dass die handler-Funktion in der in Millisekunden angegebenen Zeit periodisch bis in alle Ewigkeit und darüber hinaus aufgerufen wird. Weitere Parameter (args…) können der Funktion dabei übergeben werden.

window.requestAnimationFrame – Ich pass meine BPM an
window.requestAnimationFrame(handler) bewirkt, dass die handler-Funktion periodisch in einem für die grafische Aufbereitung sinnvoll erscheinendem Zeitintervall aufgerufen wird. Über dieses entscheidet der Browser, Chrome versucht eine Bildwiederholrate von 60 Bildern pro Sekunde zu erreichen (Frames per second, fps).

Kleine letzte Take-Aways:

- Methoden sind Klassenspezifische Funktionen, welche ohne den Aufruf function kreiert werden
- Alle zu einem Projekt gehörenden Klassen sind im Selben Namespace
- Da die Klasse jetzt in einem anderen Dokument liegt, muss sie exportiert werden