WEB TECHNOLOGIES 1

TIMERS & SYNTHETIC EVENTS

JAVASCRIPT TIMERS

- A timer is a function that enables us to execute a function at a particular time.
- Using timers you can delay the execution of code so that it does not get done at the exact moment an event is triggered or the page is loaded.
- The window object allows execution of code at specified time intervals.
- These time intervals are called timing events.
- Timer functions are implemented by browsers and their implementations will be different among different browsers.

JavaScript Timers

- There are two timer functions in JavaScript:
  - setTimeout()
  - setInterval()
- setTimeout(*function*,*milliseconds*)

  Executes a function, after waiting a specified number of milliseconds.
- setInterval(*function*,*milliseconds*)

  Same as setTimeout(), but repeats the execution of the function continuously.

SETTIMEOUT()

- The setTimeout() function is used to execute a function or specified piece of code just once after a certain period of time.
- This function accepts two parameters:
  - A *function*, which is the function to execute,
  - An optional *delay* parameter, which is the number of milliseconds representing the amount of time to wait before executing the function (1 second = 1000 milliseconds)
- Its basic syntax is

  setTimeout(*function*, *milliseconds*).

SETTIMEOUT()

- If the *delay* parameter is omitted or not specified, a value of 0 is used, that means the specified function is executed "immediately", or, as soon as possible.
- // wrong!

  setTimeout(sayHi(), 1000);
- That doesn’t work, because setTimeout expects a reference to a function. And here sayHi() runs the function, and the *result of its execution* is passed to setTimeout.
- In this case the result of sayHi() is undefined (the function returns nothing), so nothing is scheduled.

SETINTERVAL()

- setInterval() function to execute a function or specified piece of code repeatedly at fixed time intervals.
- This function also accepts two parameters:
  - *function*, which is the function to execute,
  - *interval*, which is the number of milliseconds representing the amount of time to wait before executing the function (1 second = 1000 milliseconds).
- Its basic syntax is

  setInterval(*function*, *milliseconds*).

Cancelling a Timer

- setTimeout() and setInterval() method return an unique ID i.e a positive integer value, called *timer identifier* which identifies the timer created by the these methods.
- Clearing a timer can be done using two functions:
  - clearTimeout()
  - clearInterval()
- This ID can be used to disable or clear the timer and stop the execution of code beforehand.

Synthetic Events

- A synthetic event is just an object that looks like a normal browser event,
- Also takes care of some differences in how events work between different browers.
- It has most of the (useful) fields and methods that a normal browser event has, and you can even access the 'real' browser event by looking at the nativeEvent field.

Synthetic Events

- *CustomEvent* interface is used to create your own custom event.
- And to create your own custom event, use the *CustomEvent constructor.*
- let event = new Event(type[, options]);
  - Arguments:
    - *type* – event type, a string like "click" or our own like "my-event".
    - *options* – the object with two optional properties:
      - bubbles: true/false – if true, then the event bubbles.
      - cancelable: true/false – if true, then the "default action" may be prevented. Later we'll see what it means for custom events.
  - By default both are false: {bubbles: false, cancelable: false}.

CustomEvent

- Is the same as Event.
- In the second argument (object) we can add an additional property detail for any custom information that we want to pass with the event.
- The detail property can have any data.

Creating custom events

Events can be created with the Event constructor as follows:

```
var event = new Event('build');

// Listen for the event.
elem.addEventListener('build', function (e) { ... }, false);

// Dispatch the event.
elem.dispatchEvent(event);
```

CustomEvent

- **var myEvent = new CustomEvent("myEventName");**
- Here event is going to be called **myEventName**.
- The *CustomEvent* object that wraps all of that is associated to the *myEvent* variable.
- Next step is to fire the event by using *dispatchEvent* Method.
- **var myEvent = new CustomEvent("myEventName");**
- **document.body.dispatchEvent(myEvent);**

DispatchEvent

- After an event object is created, we should “run” it on an element using the call elem.dispatchEvent(event).
- Then handlers react on it as if it were a regular browser event. If the event was created with the bubbles flag, then it bubbles.

Adding custom data – CustomEvent()

- To add more data to the event object, the CustomEvent interface exists and the detail property can be used to pass custom data.
- For example, the event could be created as follows:

```
var event=newCustomEvent('build', { detail: elem.dataset.time
  });
```

- This will then allow you to access the additional data in the event listener:
  - function eventHandler(e) {

```
    console.log('The time is: ' + e.detail);
  }
```