Telerik Academy

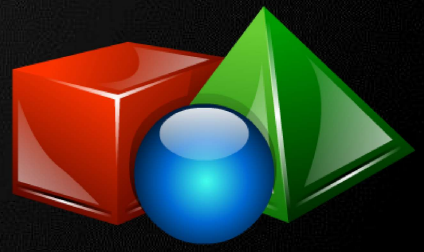
Using Objects

Objects, Properties, Primitive and Reference Types

Javascript Fundamentals

Telerik Software Academy

https://telerikacademy.com



Follow us

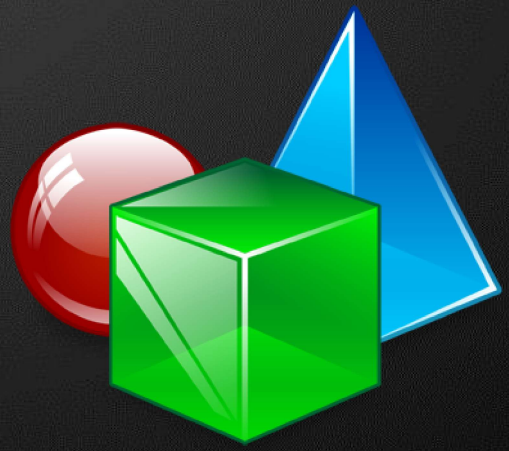
Telerik Academy

Table of Contents

Telerik Academy

Object Types and Objects

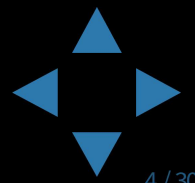
Modeling Real-world Entities with Objects



Telerik Academy

What are Objects?

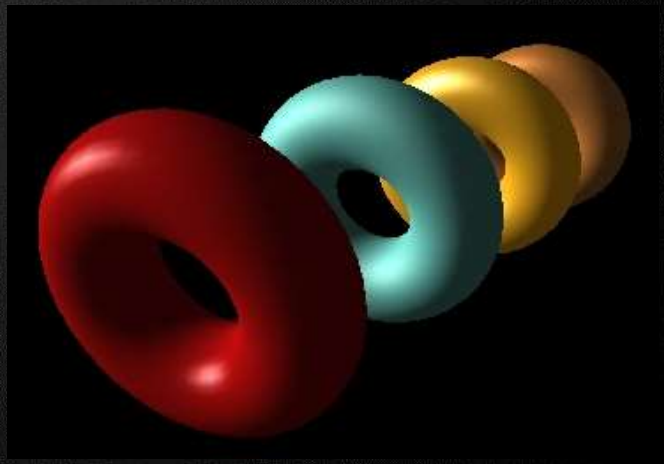
- Software objects model real-world objects or abstract concepts
  - *Examples*:
    - bank, account, customer, dog, bicycle, queue
- Real-world objects have states and behaviors
  - Account states:
    - holder, balance, type
  - Account behaviors:
    - withdraw, deposit, suspend



Telerik Academy

What are Objects?

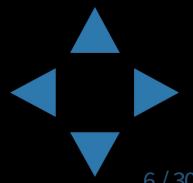
- How do software objects implement real-world objects?
  - Use variables/data/properties to implement states
  - Use methods/functions to implement behaviors
- An object is a software bundle of variables and related methods



Telerik Academy

Objects Represent

- Things from the real world
  - checks
  - people
  - shopping list
- Things from the computer world
  - numbers
  - characters
  - queues
  - arrays

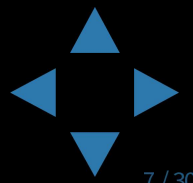


Telerik Academy

What is a Object Type?

- The formal definition of an object type:
  - Definition by Google

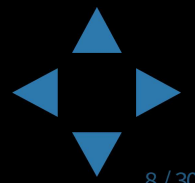
Object types act as templates from which an instance of an object is created at run time. Types define the properties of the object and the methods used to control the object's behavior.



Telerik Academy

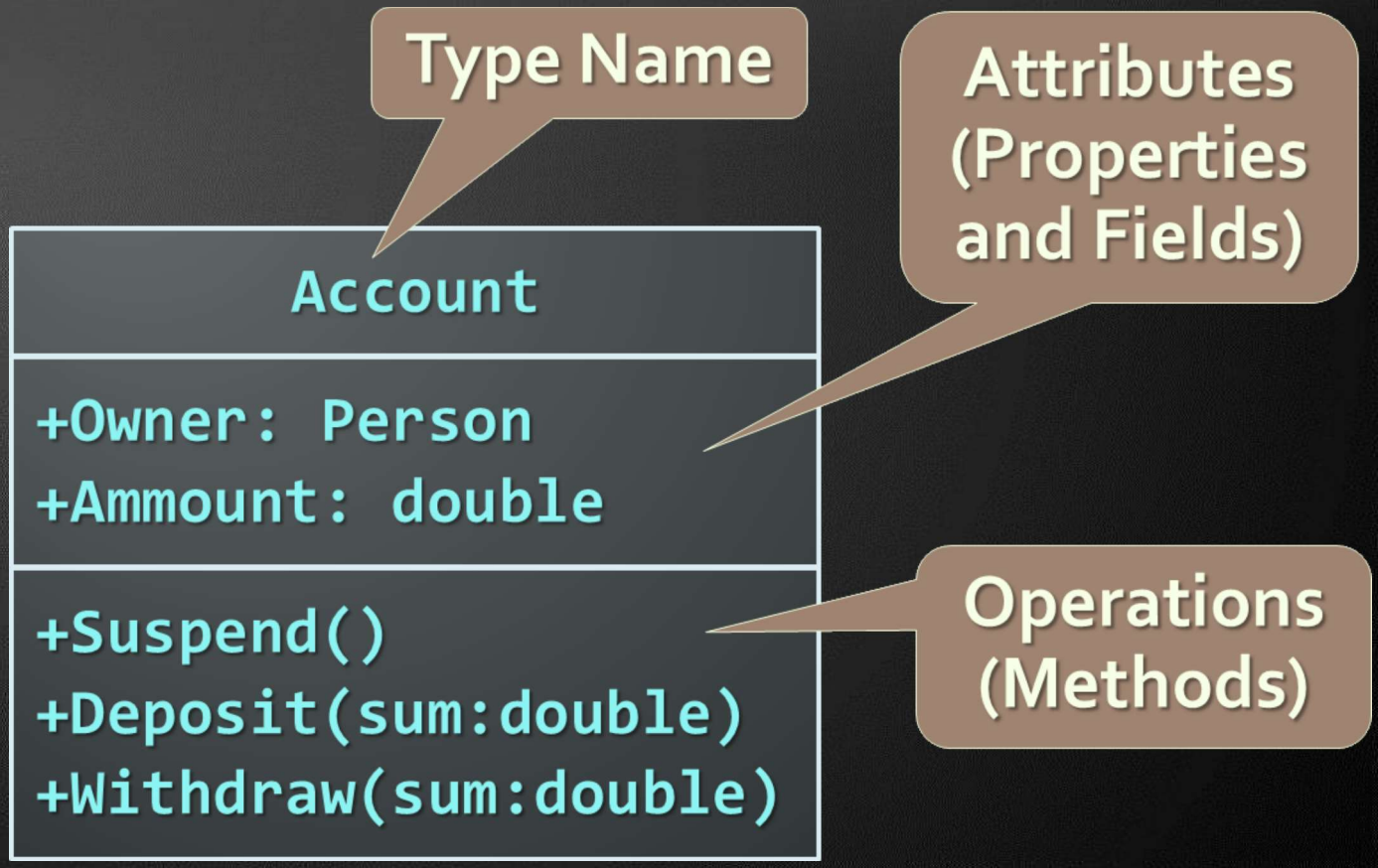
Object Types

- Object Types provide the structure for objects
  - Define their prototype, act as template
- Object Types define:
  - Set of attributes
    - Represented by variables and properties
    - Hold their state
  - Set of actions - their behavior
    - Represented by methods
- A type defines the methods and types of data associated with an object



Telerik Academy

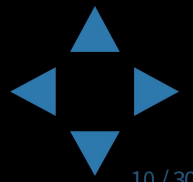
Object Types – *Example*



Telerik Academy

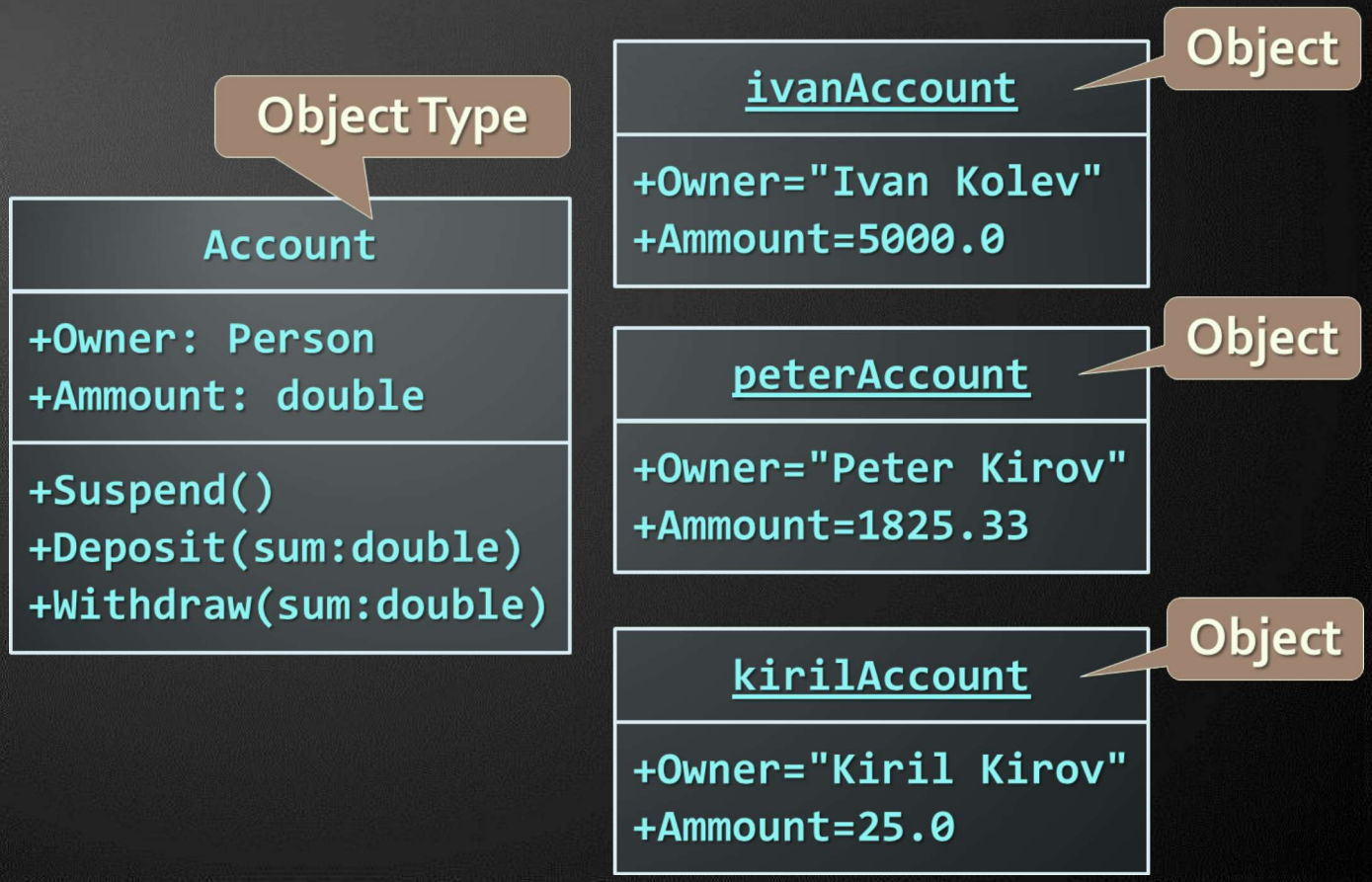
Objects

- An object is a concrete instance of a particular object type
- Creating an object from an object type is called instantiation
- Objects have state
  - Set of values associated to their attributes
- *Example*:
  - *Type:* Account
  - *Objects:* Ivan's account, Peter's account



Telerik Academy

Objects – *Example*



Telerik Academy

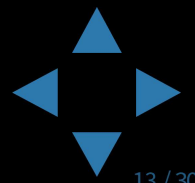
Objects

Collection of fields and methods

Telerik Academy

Objects Overview

- JavaScript is designed on a simple object-based paradigm
  - An object is a collection of properties
- An object property is association between a name and a value
  - A value of property can be either a method (function) or a field (variable)
- Lots of predefined objects available in JS
  - `Math`, `document`, `window`, etc…
- Objects can be created by the developer



Telerik Academy

Object Properties

- Each object has properties
  - Properties are values attached to the object
  - Properties of an object can be accessed with a dot-notation (`.` operator) or with `[]` - indexer:

```
let arrStr = arr.join(', '); // property join of Array
let length = arr.length;  // property length of Array
let words = text.split(' ');
let words = text['split'](' ');
```

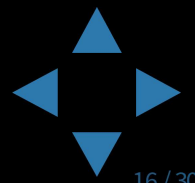

Telerik Academy

Reference and Primitive Types

Passing by value, passing by reference

Telerik Academy Reference and Primitive Types

- JavaScript is a weakly typed language
  - Variables don't have type, but their values do
- JavaScript has six different types:
  - `Number`, `String`, `Boolean`, `Null`, `Undefined` and `Object`
- `Object` is the only reference type
  - It is passed by reference (every time an object's value is used, it's used through a reference)
- `Number`, `String`, `Boolean`, `Null`, `Undefined` are primitive types
  - Passed by value (they're copied each time their value is used)



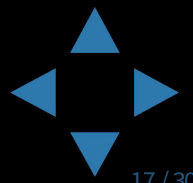
Telerik Academy

Reference and Primitive Types

- The primitive types are `Boolean`, `Number`, `String`, `Undefined` and `Null`
  - All the other types are actually of type `object`
    - Including arrays, dates, custom types, etc…

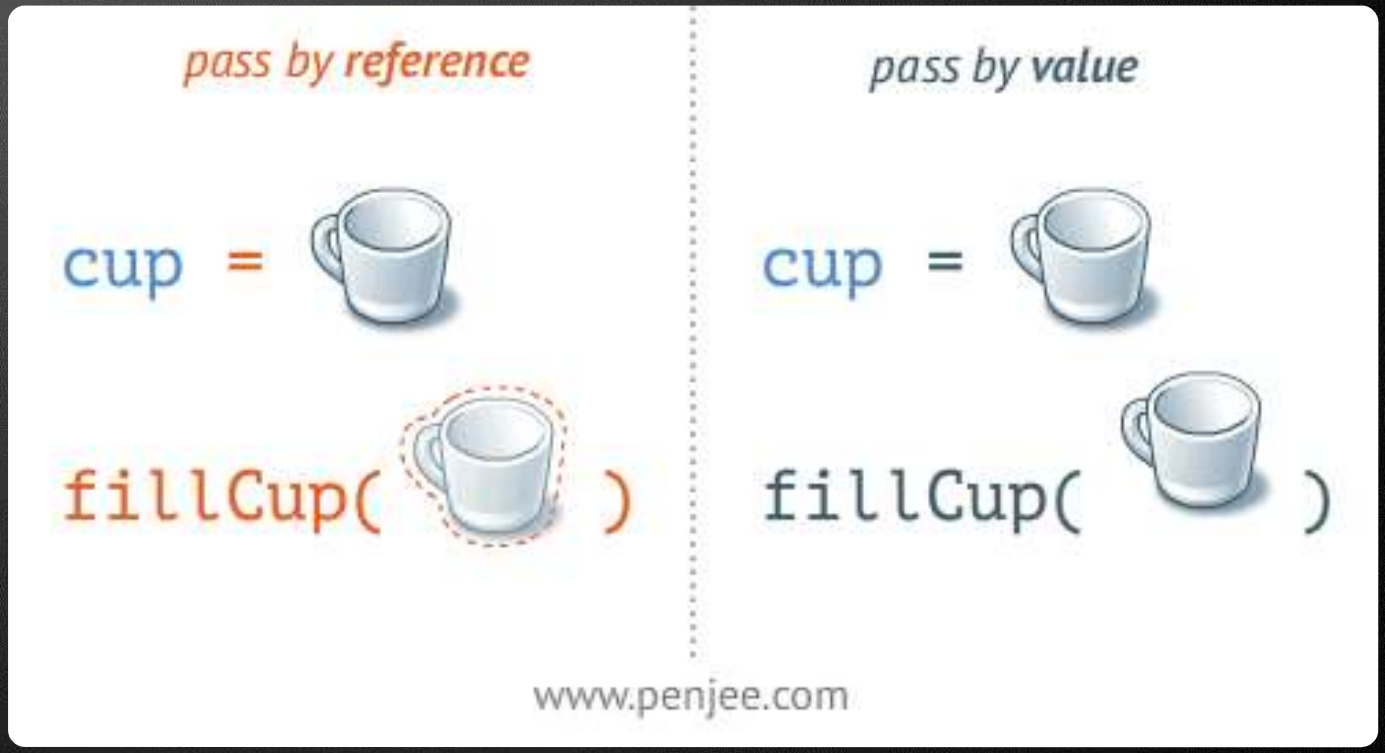
```
// all of those are true
console.log(typeof new Object() === typeof new Array());
console.log(typeof new Object() === typeof new Date());
console.log(typeof new Array() === typeof new Date());
```

- All types derive from `Object`
  - Their type is `object`



Telerik Academy

Pass by value vs. Pass by reference

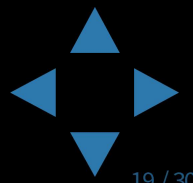


Telerik Academy

Primitive Types

- Primitive types are passed by value
  - When passed as argument
    - New memory is allocated
    - The value is copied in the new memory
    - The value in the new memory is passed
- Primitive types are initialized with type literals
- Primitive types have a object type wrapper

```
let number = 5, // Holds a primitive value of 5
    text = 'Hello there!', // Holds a primitive value
    numberObj = new Number(5); // Holds an object value of 5
```

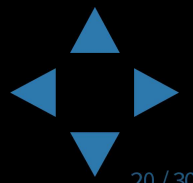



Telerik Academy

Primitive Types – *Example*

- Assign string values to two variables
  - Create an object using their value
  - Change the value of the variables
  - Each object has its own value

```
let fname = 'Peter',
    lname = 'Johnson',
    person = { firstName: fname, lastName: lname };
lname = 'Peterson';
console.log(person.lastName) // logged 'Johnson'
```

Telerik Academy

Reference Type

- `Object` is the only reference type
  - When passed it's value is used somewhere, it is not copied, but instead a reference to it is passed

```
let marks = [
  { subject : 'JavaScript', score : 4.50 },
  { subject : 'OOP', score : 5.00 },
  { subject : 'HTML5', score : 6.00 },
  { subject : 'Photoshop', score : 4.00 }];

let student = { name: 'Doncho Minkov', marks: marks };
marks[2].score = 5.50;

console.log(student.marks);
// logs 5.50 for HTML5 score
```


Telerik Academy

JavaScript Object Literal

Curly brackets { }

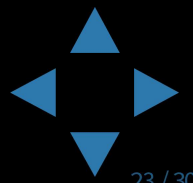
Telerik Academy

JavaScript Object Literal

- JavaScript object literal is a simplified way to create objects
  - Using curly brackets:

```
let person = {
  firstName: 'Doncho',
  lastName: 'Minkov',
  toString: function () {
    return this.firstName + ' ' + this.lastName;
  }
};

// object properties can be used:
console.log(person.toString());
// writes 'Doncho Minkov'
```

Creating Objects

- Lets make two people:

```
let minkov, georgiev;
minkov = {
  fname: 'Doncho',
  lname: 'Minkov',
  toString: function() {
    return this.fname + ' ' + this.lname;
  }
};
georgiev = {
  fname: 'Georgi',
  lname: 'Georgiev',
  toString: function() {
    return this.fname + ' ' + this.lname;
  }
};
```

- Object notations are great, but repeating code is not, right?

Telerik Academy

Object Building Function

- Using a function for building objects
  - Just pass first and last name and get a object
    - Something like a constructor

```
let minkov, georgiev;
function makePerson(fname, lname) {
  return {
    fname: fname,
    lname: lname,
    toString: function () {
      return this.fname + ' ' + this.lname;
    }
  }
}
minkov = makePerson('Doncho', 'Minkov');
georgiev = makePerson('Georgi', 'Georgiev');
```

- Much cooler, right?

Telerik Academy

JS Object Properties

Dot-notation, associative arrays

Telerik Academy

JS Object Properties

- JavaScript objects are just a set of key/value pairs
  - Each value can be accessed by its key
  - Properties in objects are accessed using the dot-notation (`obj.property`)
  - Yet properties can be used with brackets
    - Like an array

```
document.write === document['write']
```


Telerik Academy

Associative Arrays

- Objects can be used as associative arrays
  - The key (index) is string instead of number
    - Also called dictionaries or maps
- Associative arrays don't have array properties
  - `length`, `indexOf`, etc…

```
function countWords(words) {
  let word,
      wordsCount = {};
  for (let i in words) {
    word = words[i].toLowerCase();
    if (!wordsCount[word]) { wordsCount[word] = 0; }
    wordsCount[word] += 1;
  }
  return wordsCount
}
```