Inheritance and Prototypes

Class Inheritance, Prototypes, Prototype Chain, Code Reuse

SoftUni Team

Technical Trainers

Software University

http://softuni.bg

Table of Content

1. Simple Inheritance
2. Accessing Parent Members
3. Prototype Chains
4. Abstract Classes and Class Hierarchies

Have a Question?

SoftUni Foundation

sli.do

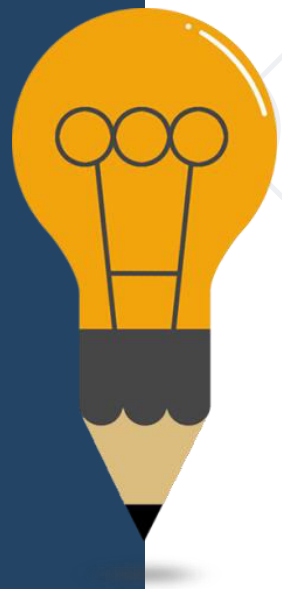
#JSCORE

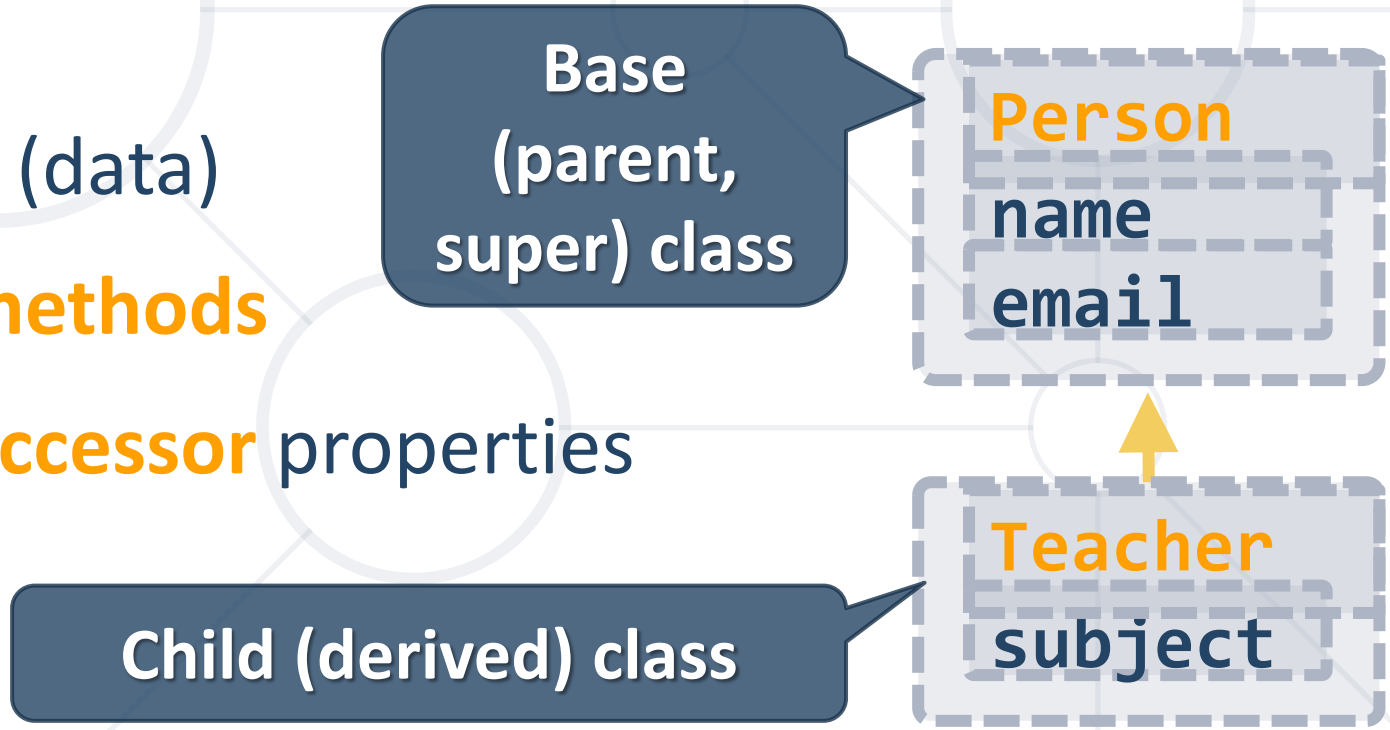
Simple Class Inheritance

Inheriting Data and Methods

Class Inheritance

- Classes can **inherit** (extend) other classes
  - Child class inherits data + methods from its parent
- **Child class** can:
  - Add **properties** (data)
  - Add / replace **methods**
  - Add / replace **accessor** properties





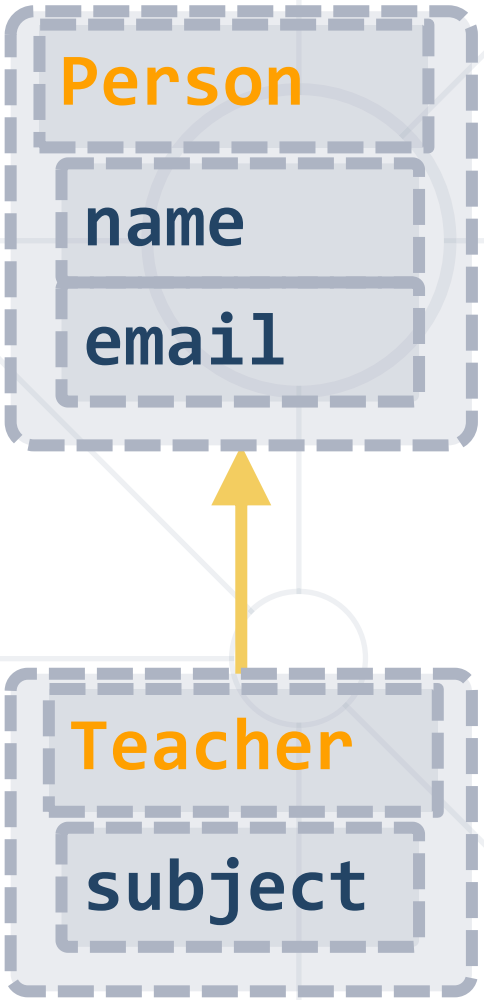
Class Inheritance – Example

```
class Person {
  constructor(name, email) {
    this.name = name; this.email = email;
  }
}
```

class Teacher inherits Person

```
class Teacher extends Person {
  constructor(name, email, subject) {
    super(name, email);
    this.subject = subject;
  }
}
```

Invoke the parent constructor



Class Inheritance – Example (2)

```
let p = new Person("Maria", "maria@gmail.com");
console.log("Person: " +
  p.name + ' (' + p.email + ')');
// Person: Maria (maria@gmail.com)
```

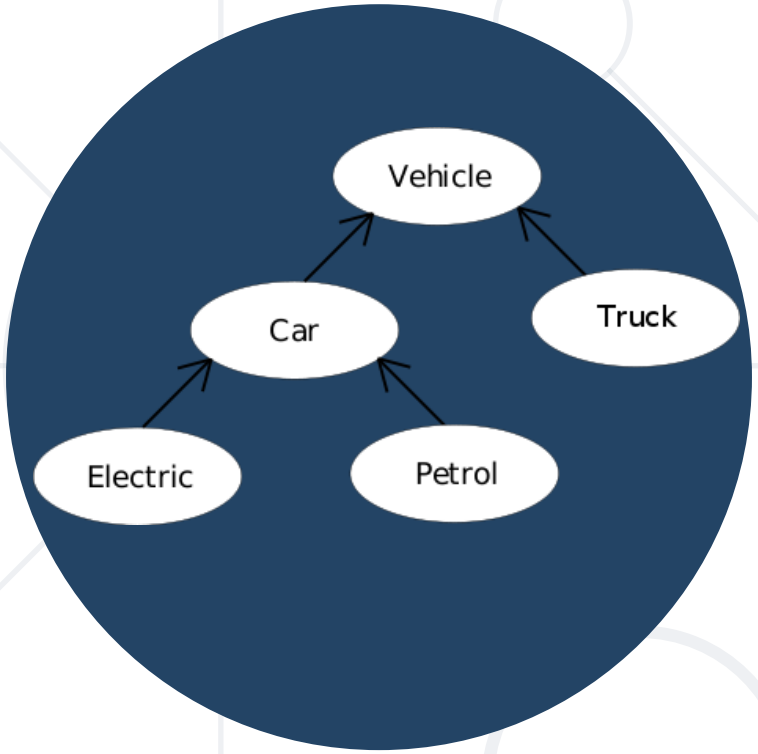
```
let t = new Teacher("Ivan", "iv@yahoo.com", "PHP");
console.log("Teacher: " + t.name +
  ' (' + t.email + '), teaches ' + t.subject);
// Teacher: Ivan (iv@yahoo.com), teaches PHP
```

Submitting Classes in the Judge

- SoftUni Judge (https://judge.softuni.bg) submissions should consist of a single function / arrow function / class definition
- Submit multiple classes as function returning JS object:

```
function personAndTeacherClasses() {
  class Person { … }
  class Teacher extends Person { … }
  return { Person, Teacher };
}
```

Check your solution here: https://judge.softuni.bg/Contests/339



Accessing Parent Members

Invoking Parent Methods

Inheriting and Replacing toString()

```
class Person {
  constructor(name, email) {
    this.name = name;
    this.email = email;
  }
  toString() {
    let className = this.constructor.name;
    return `${className} (name: ${this.name},
email: ${this.email})`;
  }
}
```

Inheriting and Replacing toString() – Teacher

SoftUni Foundation

```
class Teacher extends Person {
  constructor(name, email, subject) {
    super(name, email);
    this.subject = subject;
  }
  toString() {
    let baseStr = super.toString().slice(0, -1);
    return baseStr + `, subject: ${this.subject})`;
  }
}
```

Invoke toString() from the base (parent) class

Inheriting and Replacing toString() – Student

SoftUni Foundation

```
class Student extends Person {
  constructor(name, email, course) {
    super(name, email);
    this.course = course;
  }
  toString() {
    let baseStr = super.toString().slice(0, -1);
    return baseStr + `, course: ${this.course})`;
  }
}
```

Invoke toString() from the base (parent) class

Inheriting and Replacing toString() – Usage

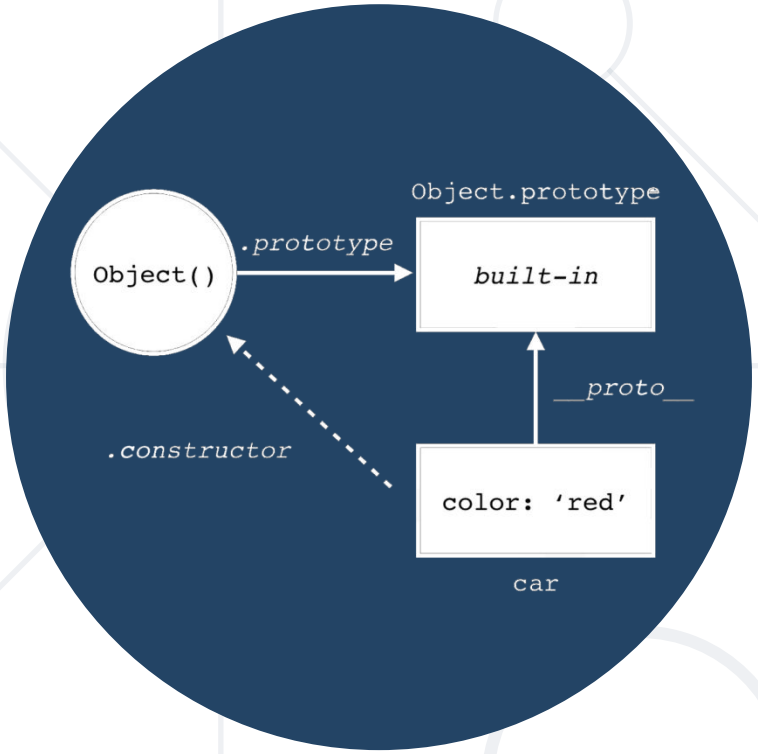
SoftUni Foundation

```
let p = new Person("Maria", "maria@gmail.com");
console.log('' + p);
// Person (name: Maria, email: maria@gmail.com)
```

```
let t = new Teacher("Ivan", "iv@yahoo.com", "PHP");
console.log('' + t);
// Teacher (name: Ivan, email: iv@yahoo.com, subject: PHP)
```

```
let s = new Student("Ana", "ana@mail.ru", 3);
console.log('' + s);
// Student (name: Ana, email: ana@mail.ru, course: 3)
```

Check your solution here: https://judge.softuni.bg/Contests/339



The Prototype Chain

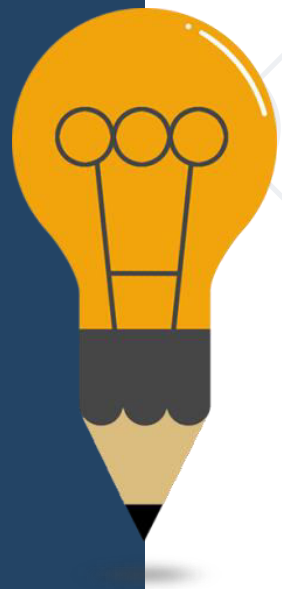
How Does It Work?

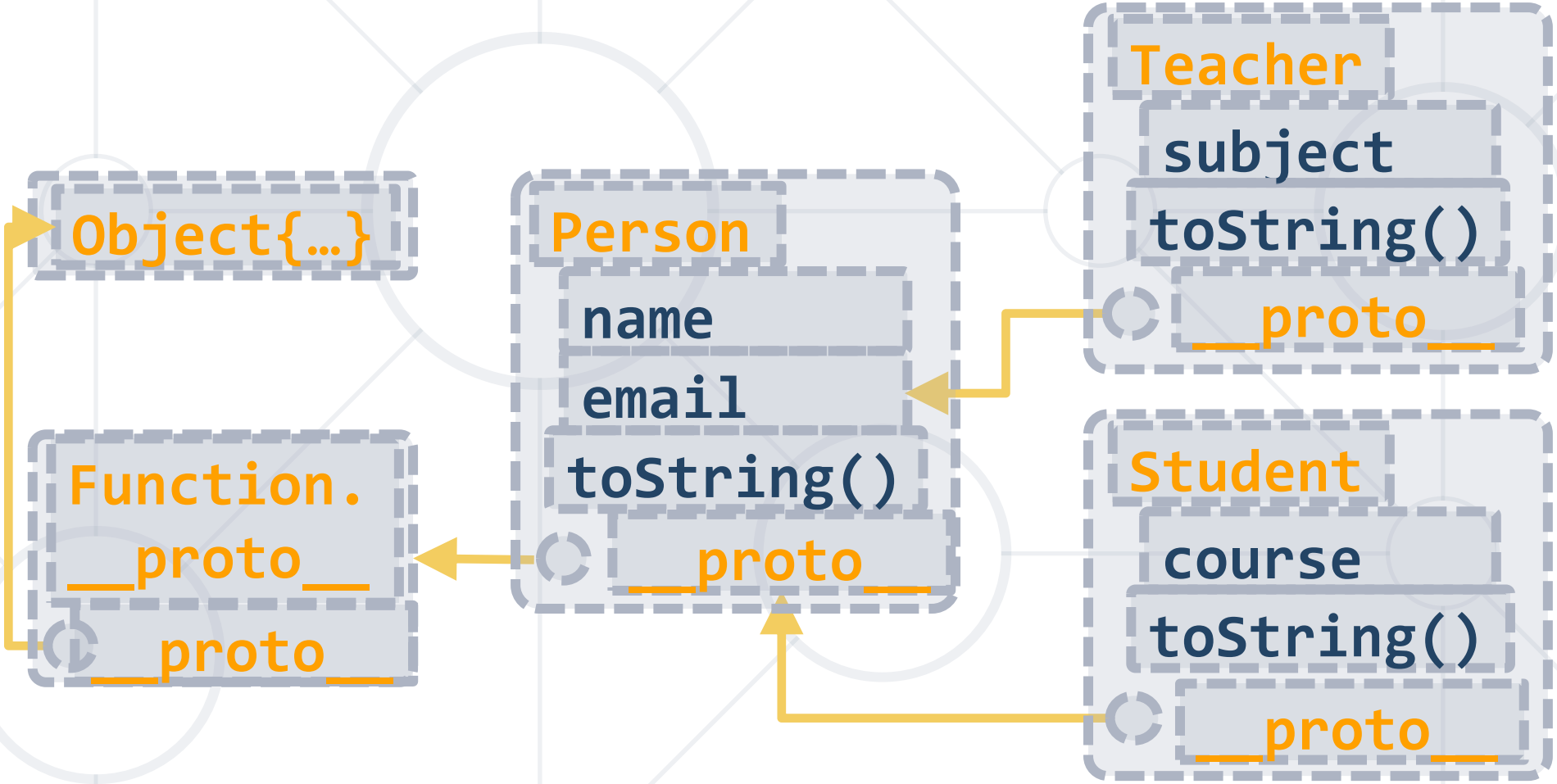
Prototypes in JavaScript

- Every object in JS has a **prototype** (template)
  - Internally called **`__proto__`** in browsers and NodeJS
  - Properties not found in the object are looked up in the prototype

- Can be obtained using **`Object.getPrototypeOf(obj)`**
- Using **`__proto__`** directly is deprecated → should be avoided!

The Prototype Chain for JS Classes





Prototype Chain (for Classes)

SoftUni Foundation

- Classes have a prototype (a parent function)
  - Prototypes form a **prototype chain**

```
Object.getPrototypeOf(Teacher) == Person
// true (class prototype holds the parent class)
```

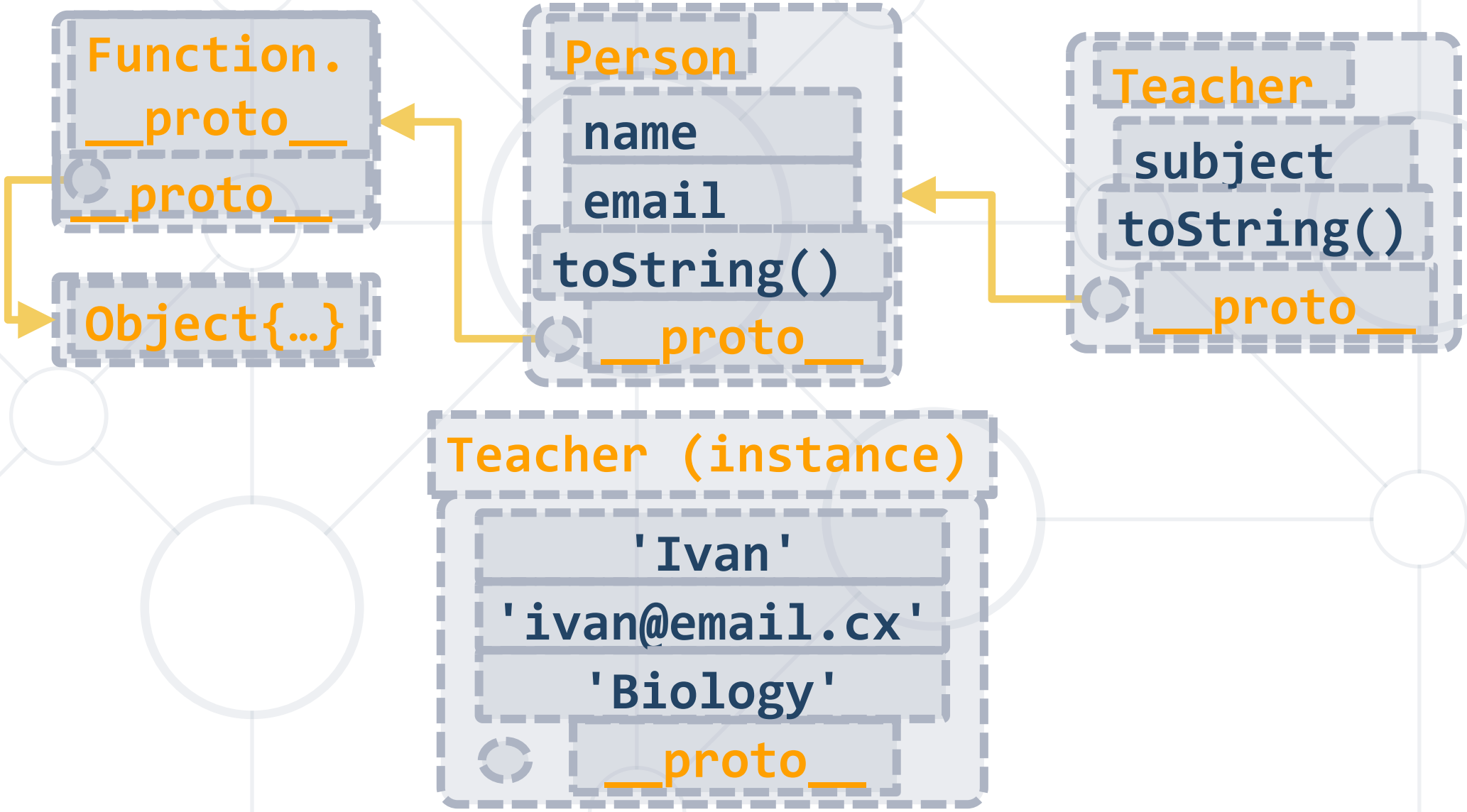
```
Teacher.__proto__ == Person; // true
// The same as the above (unofficial property)
```

```
Object.getPrototypeOf(Person) == Function.prototype; // true
```

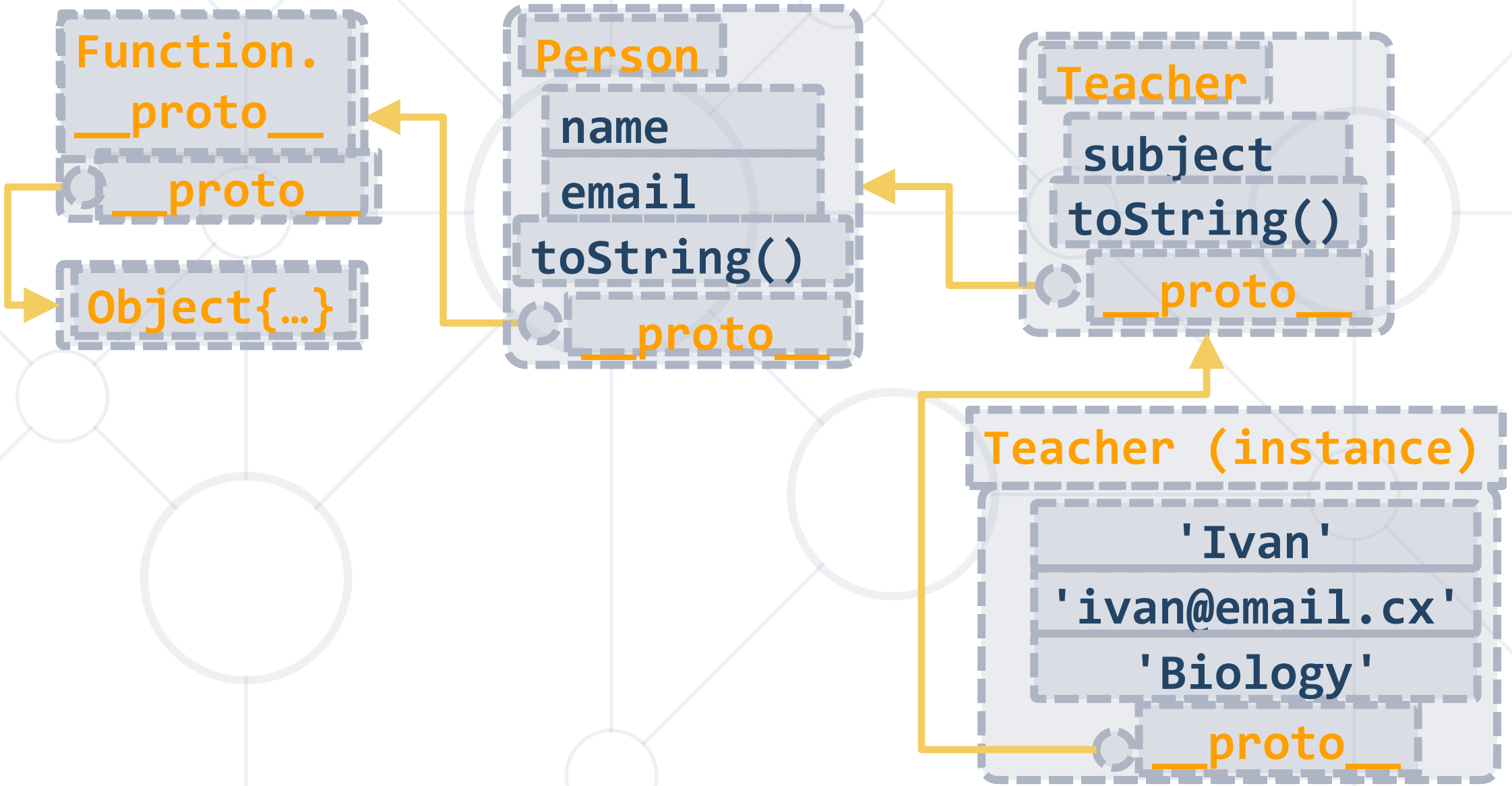
Prototypes in Classes and Objects

- Prototypes in **classes** / functions
  - Classes use their **__proto__** to resolve methods / properties
    - **__proto__** holds the parent class / function
  - Classes have also **prototype** object used to create new objects
    - Assigned to **__proto__** of each new object
- Prototypes in **objects**
  - Objects use their **__proto__** to resolve methods / properties
  - Objects do not have **prototype** object

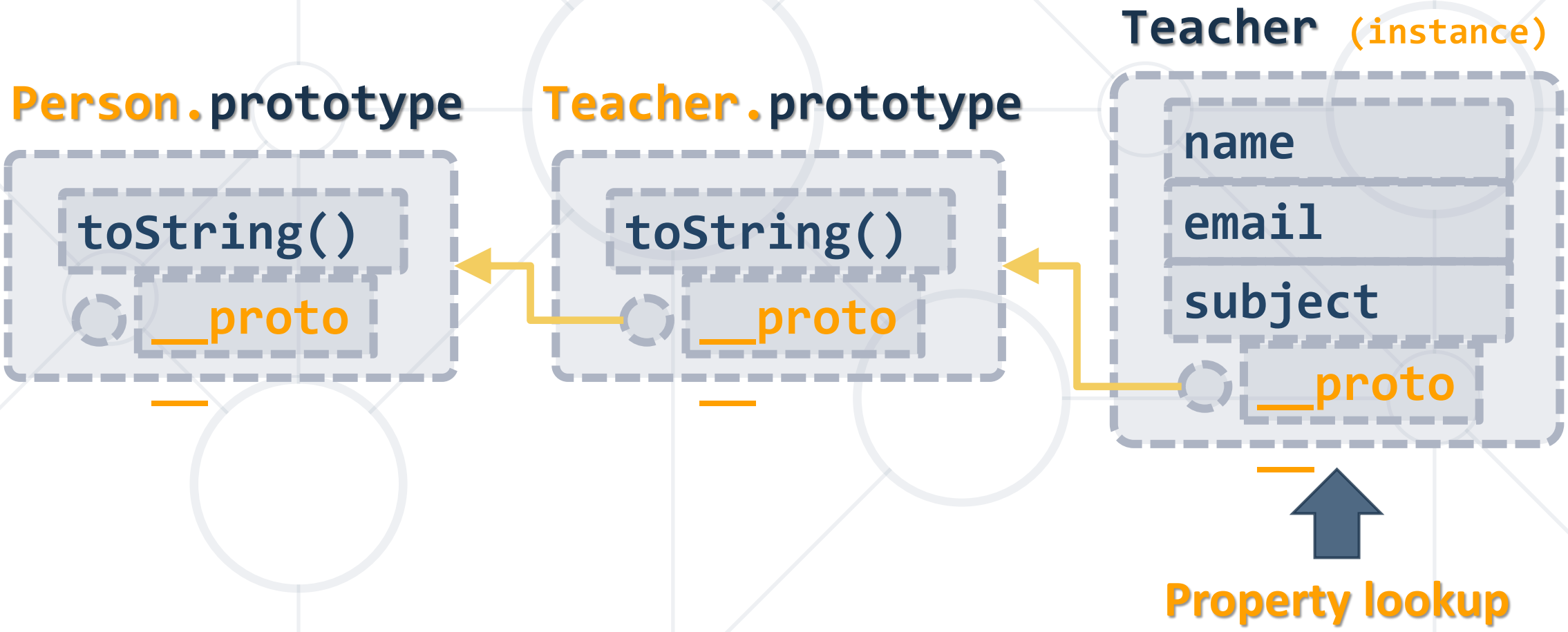
Object Instantiation (Create New Object)



Object Instantiation (Create New Object)



Prototype Chain for JS Objects



Problem: Extending Prototype

- Extend a passed class's **prototype** with a property **species** and m ethod **toSpeciesString()**:
  - **Person.prototype.species** – holds a string value "*Human*"
  - **Person.prototype.toSpeciesString()** – returns "I am a **{species}. {class.toString()}**"

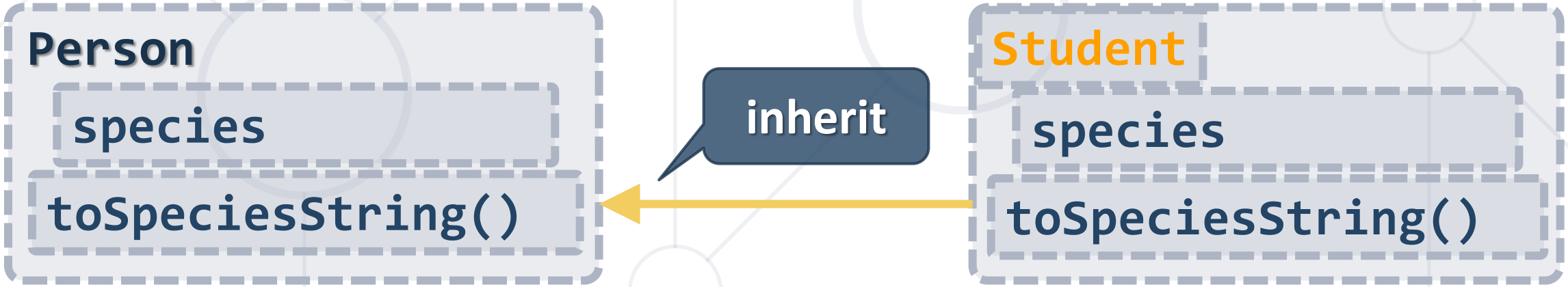
```
new Person("Maria", "maria@gmail.com").toSpeciesString()
// "I am a Human. Person (name: Maria, email: maria@gmail.com)"
```

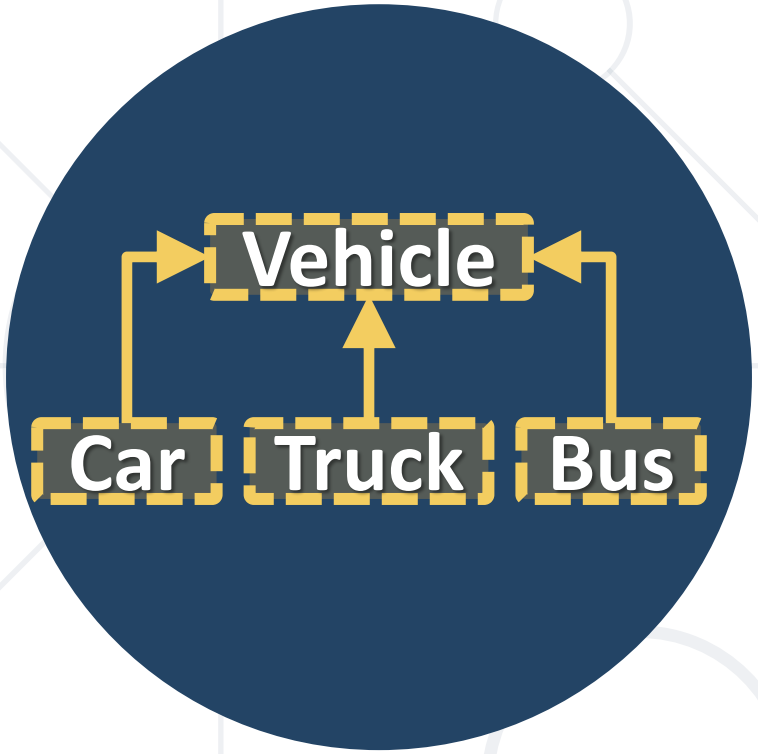
```
new Student("Ana", "ana@mail.ru", 3).toSpeciesString()
// "I am a Human. Student (name: Ana, email: ana@mail.ru, course: 3)"
```

Solution: Extending Prototype

```
function extendPrototype(Class) {
  Class.prototype.species = "Human";
  Class.prototype.toSpeciesString = function () {
    return `I am a ${this.species}. ${this.toString()}`;
  }
}
```

```
extendPrototype(Person);
```





Abstract Classes and Mixins

Inheriting Pieces of Functionality

What is Abstract Class?

- **Abstract classes** are abstractions → cannot be instantiated
  - Check **new.target** in the constructor

```
class Abstract {
  constructor() {
    if (new.target === Abstract)
      throw new TypeError("Cannot
construct Abstract instances directly");
  }
}
```

What is a Mixin?

- **Mixins** are bits of functionality that can be added to objects of **different** classes
  - Allow extending existing classes without modifying t hem directly
  - The code becomes more portable

```
function Mixin() {
  this.extensionFunc = function() { // New functionality …};
  return this;
}
```

What is a Mixin?

```
function asCircle() {
  this.area = function() {
    return Math.PI * this.radius * this.radius;
  };return this;
}
class Circle {constructor(r) { this.radius = r; } }
asCircle.call(Circle.prototype);
let circle = new Circle(5);
circle.area();
```

asCircle

area()

Circle

radius

Checking the Object Type

- Checking object type using `constructor.name`

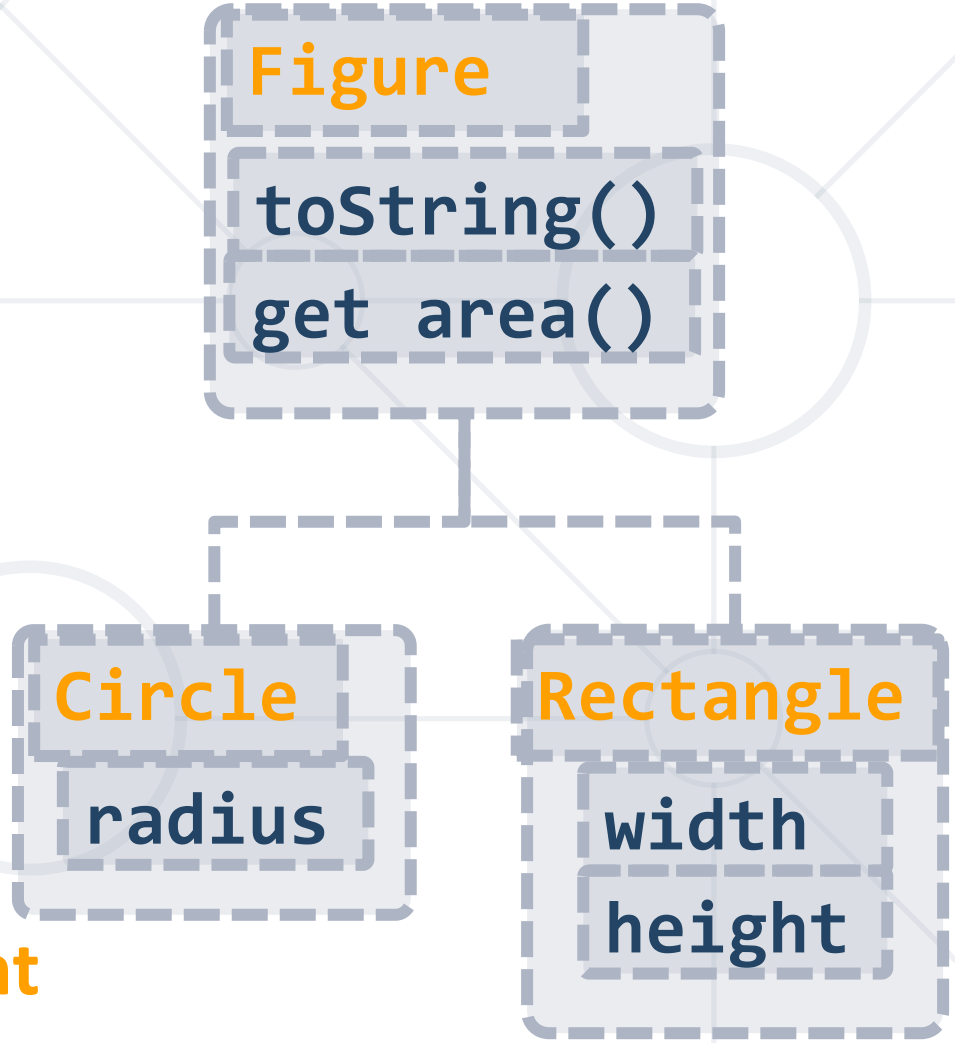
```
let p = new Person("Pesho","pesho@hit.bg");
console.log(p.constructor.name);
// Person
```

- Check if object belongs to a certain class (or its descendant):

```
let t = new Teacher("Pesho","pp@hit.bg","PHP");
console.log(t instanceof Person)
// true
```

Problem: Class Hierarchy

- Define the following class hierarchy:
  - **Figure**
    - **Abstract** class, defines **toString()** and **get area()**
  - **Circle**
    - Extends **Figure**, adds **radius**
  - **Rectangle**
    - Extends **Figure**, adds **width** + **height**



Solution: Class Hierarchy

SoftUni Foundation

```
function classHierarchy() {
  class Figure {
    constructor() {
      if (new.target === Figure)
        throw new Error("Cannot instantiate an abstract class.");
    }
    get area() { return undefined; }
    toString() { let type = this.constructor.name; return type;
    }
  }
```

abstract method → will be implemented in child classes

Solution: Class Hierarchy (2)

```
class Circle extends Figure {
  constructor(radius) {
    super();
    this.radius = radius;
  }
  get area() { return Math.PI * this.radius * this.radius; }
  toString() {
    return super.toString() + ` - radius: ${this.radius}`;
  }
}
```

Solution: Class Hierarchy (3)

```
  class Rectangle extends Figure {
    constructor(width, height) {
      super(); [this.width, this.height] = [width, height];
    }
    get area() { return this.width * this.height; }
    toString() {
      return super.toString() + ` - width: ${this.width}, height
: ${this.height}`;
    }
  } return { Figure, Circle, Rectangle };
}
```

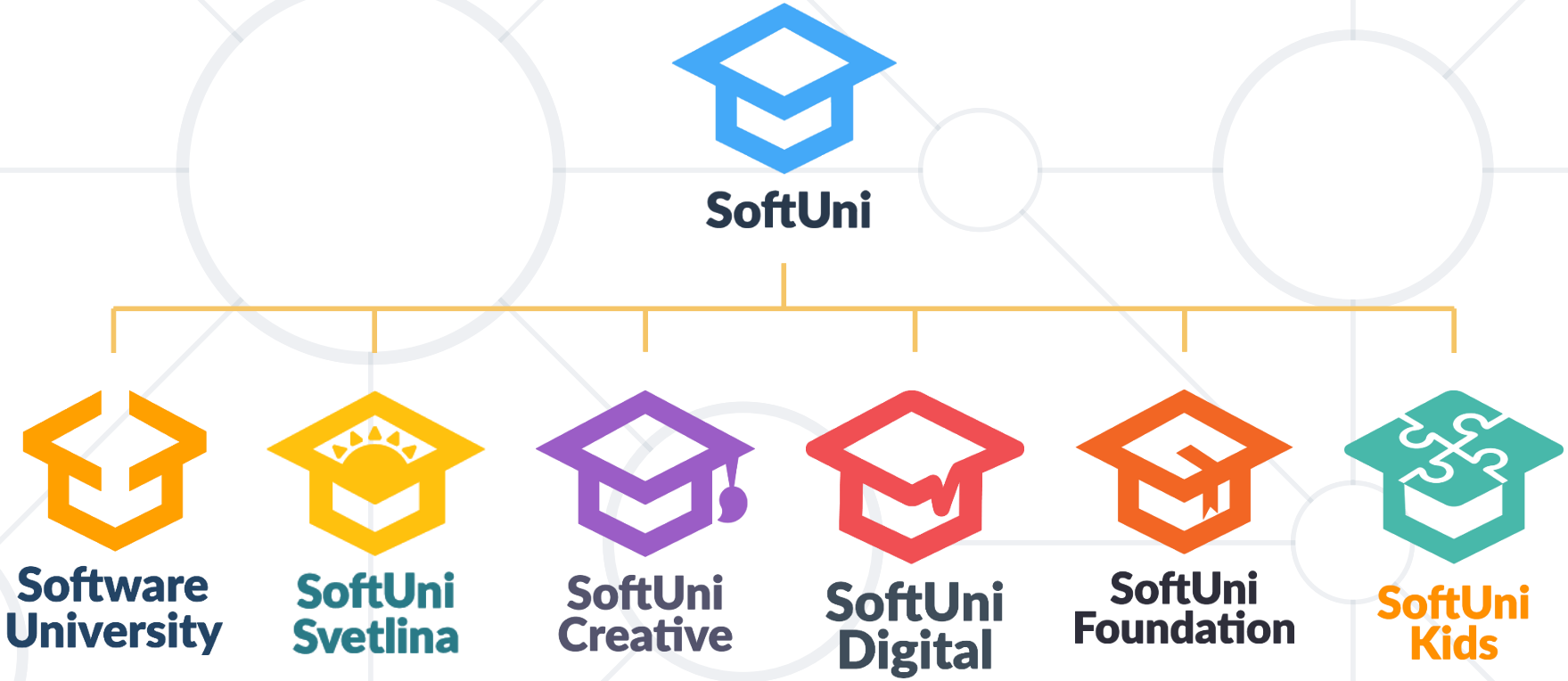
Check your solution here: https://judge.softuni.bg/Contests/339

Live Exercises

Summary

- Inheritance allows **extending** existing classes
- Objects in JS have **prototypes**
  - Objects look for **properties** in their prototype chains
  - Prototypes form a **hierarchical chain**

Questions?



https://softuni.bg/trainings/2081/js-advanced-october-2018

SoftUni Diamond Partners

SoftUni Foundation

XSsoftware

SBTech
we know sports

telenor

SoftwareGroup
doing it right

NETPEAK

SmartIT

Postbank
Решения за твоето утре

SUPER HOSTING .BG

INDEAVR
Serving the high achievers

INFRAGISTICS

LIEBHERR

æternity

codexio

SoftUni Organizational Partners

SoftUni Foundation

ИНФОРМАЦИОННО ОБСЛУЖВАНЕ

OneBit SOFTWARE

WORLD OF MYTHS

Lukanet.com

codeXio

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education and Employment Opportunities
  - softuni.bg
- Software University Foundation
  - http://softuni.foundation/
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License

- This course (slides, examples, demos, videos, homework, etc.) is licensed under the "Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International" license