Arv

Återanvändning av kod

Upprepning av kod

- ✓ Klasserna `Person` och `Student` innehåller kod gemensam för de båda typerna.
  - Behöver det vara så?
  - Nej, med hjälp av arv (*"inheritance"*) kan kod återanvändas istället för att upprepas.

```
// person.js
export class Person {
  constructor (name, age) {
    this.name = name
    this.age = age
  }

  toString () {
    return `Jag, ${this.name}, är ${this.age} år.`
  }
}
```

Exakt samma kod!
(Inte sååå fruktansvärt, men vi måste börja någonstans.)

Metod som ger textbeskrivning av ett objekt.

```
// student.js
export class Student {
  constructor (name, age, onCampus) {
    this.name = name
    this.age = age
    this.onCampus = onCampus
  }

  toString () {
    const form = this.onCampus ? 'campus' : 'distans'
    return `Jag, ${this.name}, läser på ${form}, och är ${this.age} år.`
  }
```

Återanvändning av kod genom arv

- ✓ Genom att låta `Student` ärva från `Person` kan kod i `Person` återanvändas av `Student`.

```
// person.js
export class Person {
  constructor (name, age) {
    this.name = name
    this.age = age
  }

  toString () {
    return `Jag, ${this.name}, är ${this.age} år.`
  }
}
```

Då `Student` ärver från `Person` sägs `Person` vara basklass till `Student`.

`Student` ärver från `Person` och utökar basklassen.

```
// student.js

import { Person } from './person.js'

export class Student extends Person {
  constructor (name, age, onCampus) {
    super(name, age)
    this.onCampus = onCampus
  }

  toString () {
    const form = this.onCampus ? 'campus' : 'distans'
    return `Jag, ${this.name}, läser på ${form}, och är ${this.age} år.`
  }
}
```

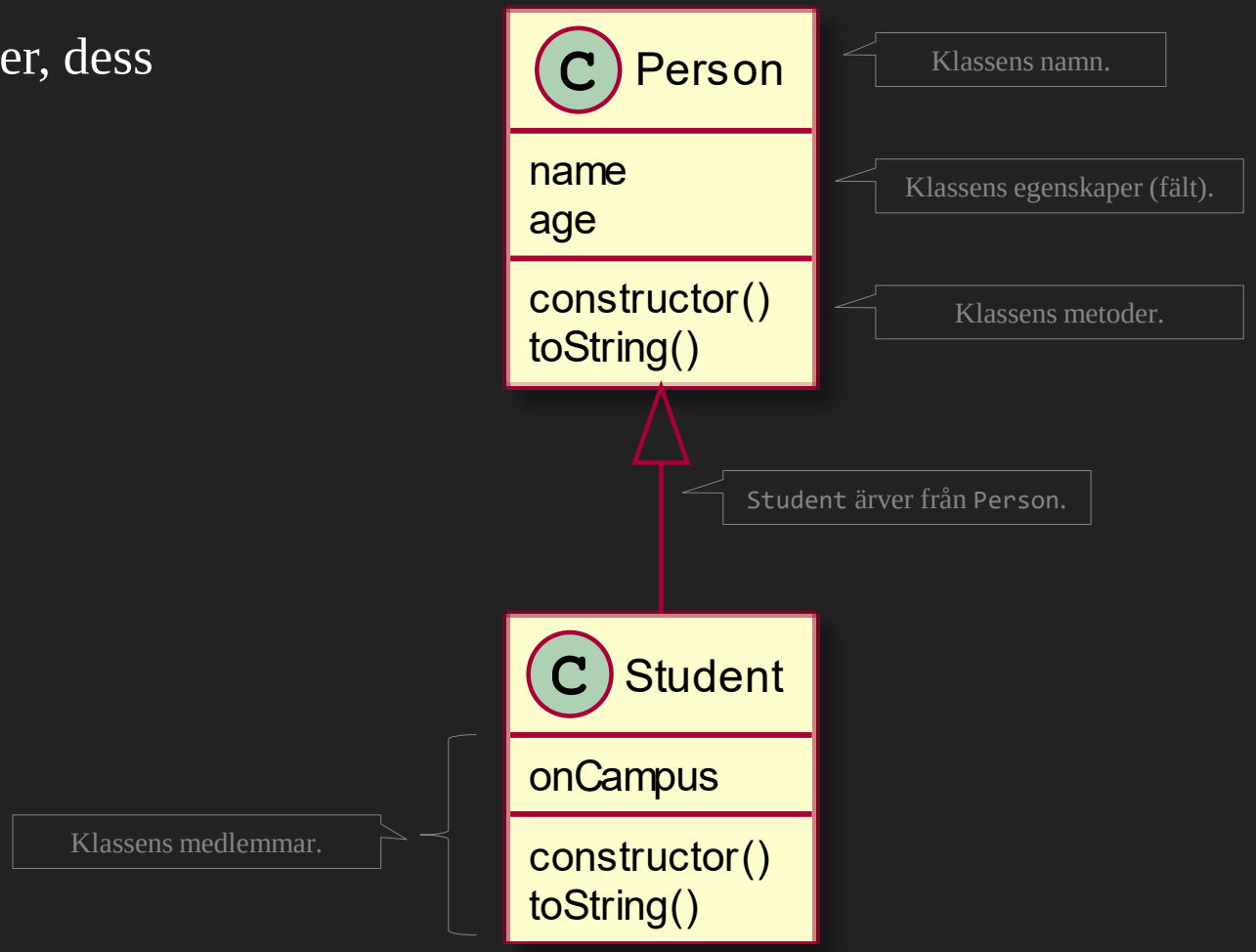
Metoden `super()` anropar basklassens konstruktor för att påbörja initieringen av det nya objektet.

Metoden `toString()` i `Student` överskuggar (”overrides”) implementationen av `toString()` i `Person`.

Klassdiagram

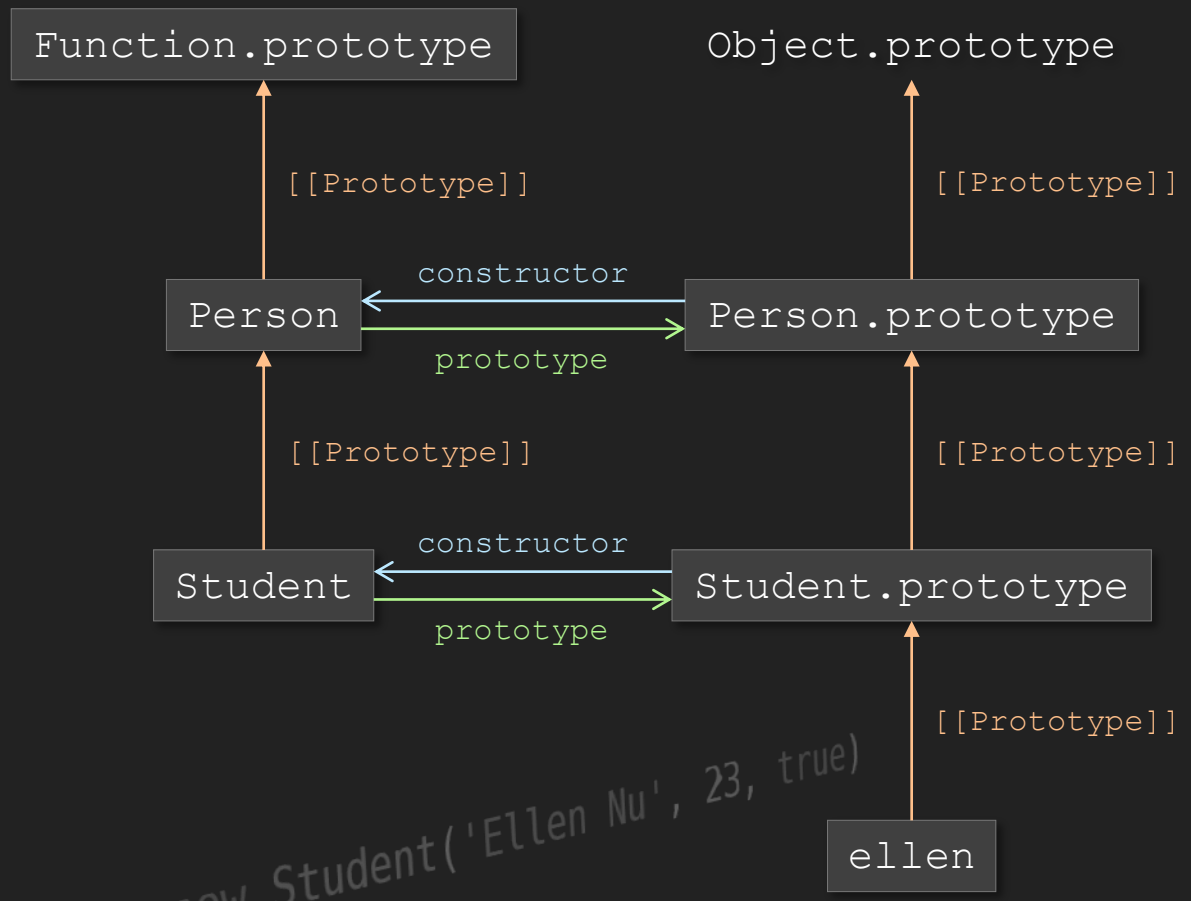
- ✓ Ett klassdiagram beskriver bland annat klasser, dess medlemmar och arv.

```
// person.js
export class Person {
  constructor (name, age) {
    this.name = name
    this.age = age
  }

  toString () {
    return `Jag, ${this.name}, är ${this.age} år.`
  }
}

// student.js
import { Person } from './person.js'

export class Student extends Person {
  constructor (name, age, onCampus) {
    super(name, age)
    this.onCampus = onCampus
  }

  toString () {
    const form = this.onCampus ? 'campus' : 'distans'
    return `Jag, ${this.name}, läser på ${form}, och är ${this.age} år.`
  }
}
```



Arv under huven – en massa objekt

- Genom att låta `Student` ärva från `Person` skapas flera objekt under huven och två prototypkedjor byggs.
  - Till höger är instansprototypkedjan.
  - Till vänster är klassprototypkedjan.
- Instansprototypkedjan börjar med `ellen`, för att fortsätta med `Student.prototype`, därefter med `Person.prototype` och avslutas med `Object.prototype`.
- Klassprototypkedjan börjar med `Student` som följs av `Person`, som i sin tur följs av `Function.prototype` (eftersom `Person`, och därmed `Student`, är en funktion).



Av vilken typ är ett objekt?

- Med operatorn `instanceof` kan du undersöka om ett värde är en instans av en given klass.

```
// app.js
import { Person } from './person.js'
import { Student } from './student.js'

const person = new Person('Nisse', 42)
const student = new Student('Ellen', 23, true)

console.log(person instanceof Person) // OUTPUT: true
console.log(person instanceof Student) // OUTPUT: false

console.log(student instanceof Person) // OUTPUT: true
console.log(student instanceof Student) // OUTPUT: true
```

Nisse är ingen student!

Ellen är en student, som är en person.