Arv, Subklasser, super, Polymorfism

Arv: En klass baserad på en annan klass

Föräldraklass och Subklass

Det går att deklarera en klass som baseras på en annan klass:

```
class Person {
  constructor (name) {
    this.name = name;
  }
  sayHi () {
    console.log(`Hi, my name is ${this.name}`);
  }
}

class Student extends Person {
  constructor (name, university) {
    super(name);
    this.university = university;
  }
}

let s1 = new Student("Janis", "MaU");
s1.sayHi(); // loggar "Hi, my name is Janis"
```

Föräldraklassen (Parent class)

Nyckelordet *extends* markerar att denna klass baseras på en annan, och är därför en **subklass**

super representerar föräldraklassen

Arv: instanser

Subklassens instanser ärver metoder och egenskaper från föräldraklassen

```
class Person {
  constructor (name) {
    this.name = name;
  }
  sayHi () {
    console.log(`Hi, my name is ${this.name}`);
  }
}

class Student extends Person {
  constructor (name, university) {
    super(name);
    this.university = university;
  }
}

let s1 = new Student("Janis", "MaU");
s1.sayHi(); // loggar "Hi, my name is Janis"

console.log(s1.name);  // Janis
```

Arv: statiska metoder och egenskaper

Subklassen ärver statiska metoder och egenskaper från föräldraklassen

```
class Person {
  static property = 42;
  static method () { console.log("static Person"); }
}

class Student extends Person {}

console.log(Student.property); // 42
Student.method(); // loggar "static Person"
```

Polymorfism

Ibland vill man att samma metod ska göra olika saker i olika subklasser (underklasser).

Exempel: Ett spel där man programmerar ett antal olika djur, som alla kan "makeSound", men det ska hända olika saker när olika djur gör det.

```
class Animal {
  makeSound () { console.log("Sound!"); }
}

class Cat extends Animal {
  makeSound () { console.log("Mjau!"); }
}

class Worm extends Animal {}

const cat1 = new Cat();
cat1.makeSound(); // Mjau!

const worm1 = new Worm();
worm1.makeSound(); // Sound!

const animal1 = new Animal();
animal1.makeSound(); // Sound!
```

Polymorfism: Mer abstraherat alternativ

super representerar föräldraklassen även i metodernas "body"

```
class Animal {
  makeSound (what = "Sound!") { console.log(what); }
}

class Cat extends Animal {
  makeSound () { super.makeSound("Mjau!"); }
}

class Worm extends Animal {
  makeSound () {}
}

const cat1 = new Cat();
cat1.makeSound(); // Mjau!

const worm1 = new Worm();
worm1.makeSound(); // nothing happens

const animal1 = new Animal();
animal1.makeSound(); // Sound!
```

Polymorfism: Fungerar också för egenskaper

```
class Animal {
  constructor (nLegs = 4) {
    this.nLegs = nLegs;
  }
}

class Cat extends Animal {}
```

```
class Ant extends Animal {
  constructor () {
    super(6);
  }
}

const cat1 = new Cat();
cat1.nLegs; // 4

const ant1 = new Ant();
ant1.nLegs; // 6
```

Polymorfism: Fungerar också för static

```
class Animal {
  static nLegs = 4
  static m () { console.log("Animal Static"; }
}

class Cat extends Animal {
 static m () { console.log("Cat Static"; }
}

class Ant extends Animal {
  static nLegs = 6
  static m () { console.log("Ant Static"; }
}

Animal.nLegs; // 4
Cat.nLegs; // 4
Ant.nLegs; // 6
```