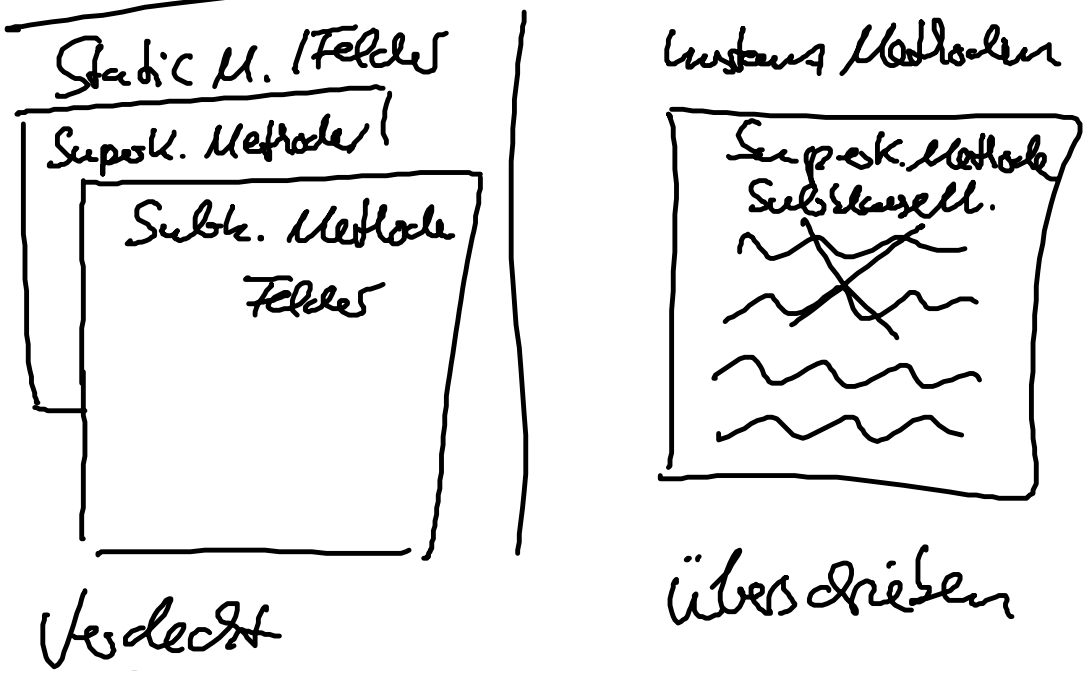
```
class FlyMonster { ... }
class Dragon extends FlyMonster { ... }

Dragon A = new Dragon();
FlyMonster B = A;

Dragon.  testClassMethod();   // -> Methode der Subklasse
Monster. testClassMethod();   // -> Methode der Basisklasse

A. testInstanceMethod();
B. testInstanceMethod();      // -> Beide sagen: ich bin ein Drache!
```

verdeckt

überschreiben

Verdecken & überschreiben



Grundsätzlich gilt:

Überschreiben ist cool!

Verdecken ist blöd!

Verwendung von Super

```
public class FlyMonster{
  public void greet() { Sy...("Here is classFM");}
}
public class Dragon extends FlyMonster{
    public void greet() {
      super.greet();
      Sy... ("This is class Dragon.");
    }
}
```

```
Dragon A = new Dragon();
    A.greet() -> This is class FlyMonster
                 This is " " Dragon
```

```
public Dragon ( int weight, int numberOfW.
                int speed ) {
    super( weight, speed ); // ruft den Konstr.
                            // FlyMonster(int,int)
                            // auf
    this.numberOfWings = numberOfWings;
}
```

-> super() wird von Java sowieso gemacht!

-> deshalb muss jede Superklasse einen default Kon. haben.

-> im Konstruktor verwendete Elemente (Methoden, Felder, Klassen) sollten nicht überschrieben werden!!! Also final setzen.

- Klasse die final ist kann nicht abgeleitet!

Die Superklasse Object

- protected Object clone() throws CloneNotSupportedException

- public boolean equals (Object obj) kann nicht überschrieben werden!

- public (final) Class getClass ()
  -> gibt ein Objekt vom Typ Class

z.B. Obj.getClass().getSimpleName()
.getSuperclass()
↗ was ist das für Objekt

- public String toString()
- public int hashCode() (Speicheradresse des Objekts)

und finalize(), notify(), notifyAll(), mehrere wait(-) Methoden

„Das Gegenteil von final":

Abstract

- Ein Methode die abstract ist muss überschrieben werden!

```
abstract class Schachfigur {                ← einzelne ..
    :
    abstract void moveTO (Schachfeld Ziel);
                                            ↑ kein {..}
    :
}
class Springer extends Schachfigur {
    :
    void moveTO (Schachfeld Ziel) {... }
    :
```