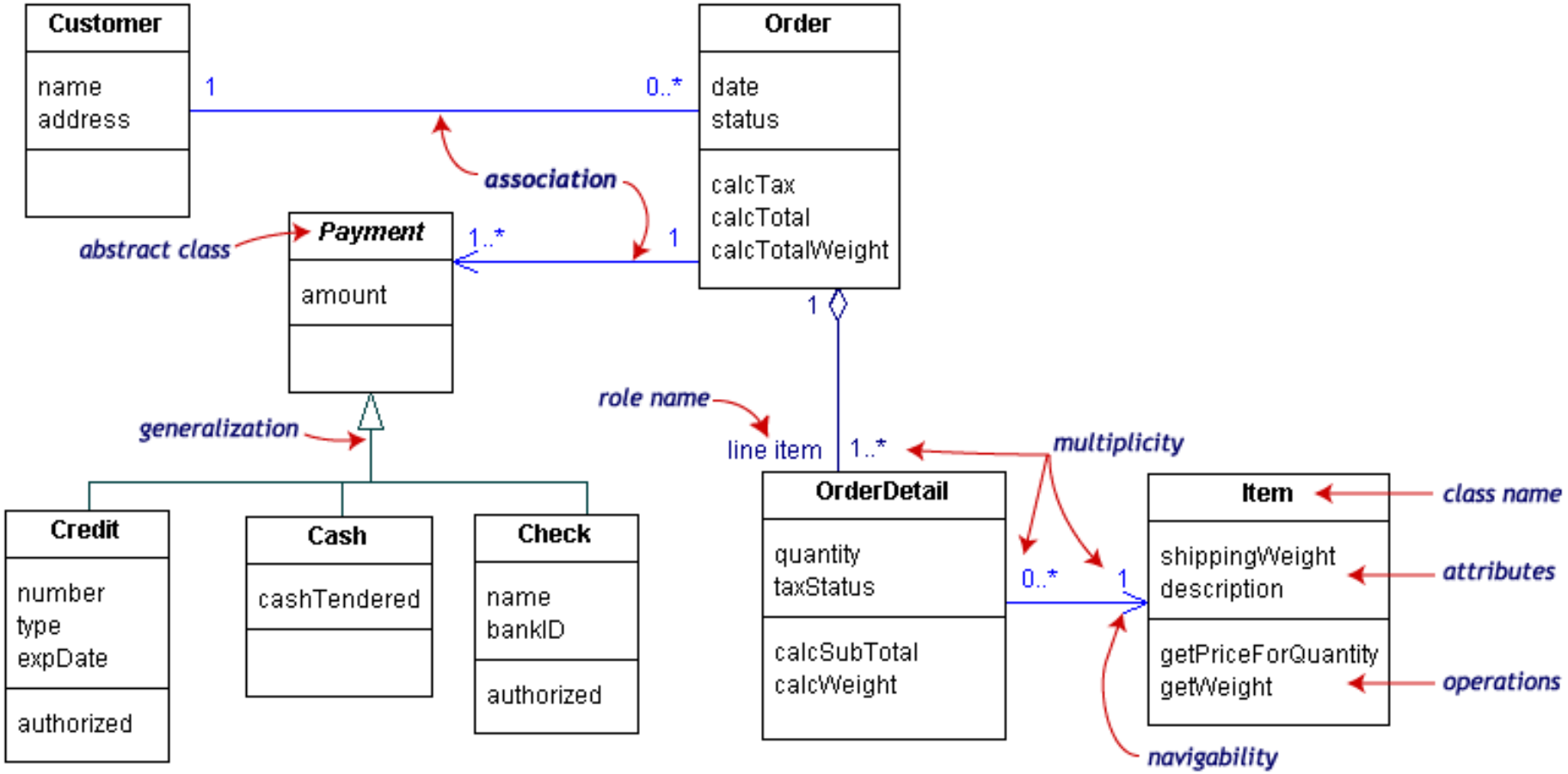
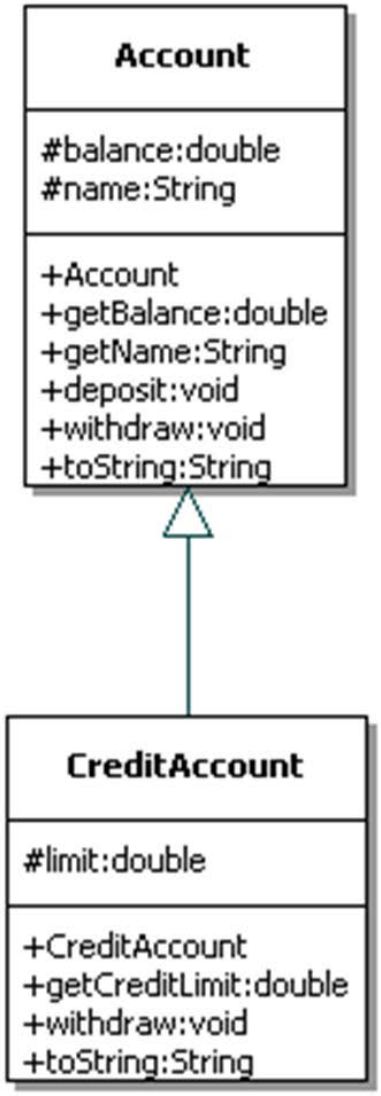
Arv, grunder

Relationer mellan *objekt* – association, aggregat, komposition



Källa: http://edn.embarcadero.com/article/images/31863/classdiagramno3d.gif

Arv – relation mellan *klasser*, ”är en”



Java-syntax:

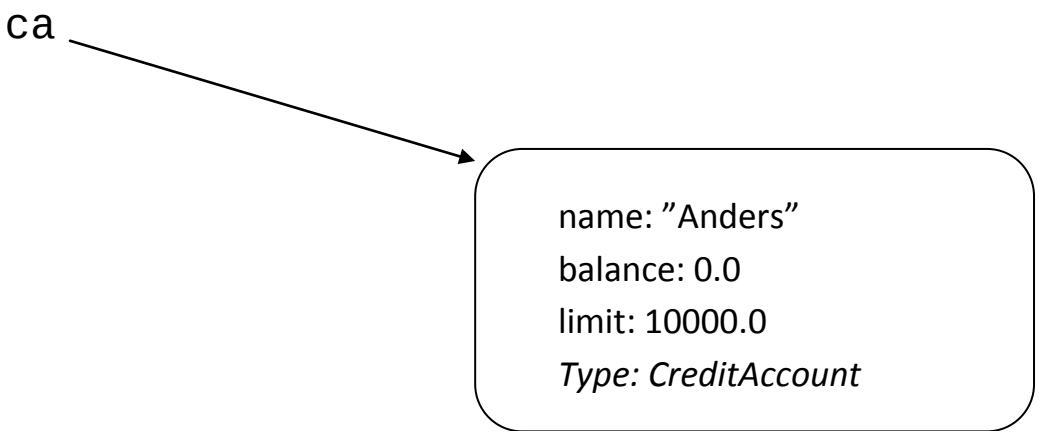
```
public class CreditAccount extends Account {
…
```

Skapa objekt på vanligt sätt:

```
CreditAccount ca =
   new CreditAccount("Anders", 10000.0);
```

Arv, ”är en”

```
CreditAccount ca =
   new CreditAccount("Anders", 10000.0);
```



Synlighet, visibility, i Java

Keyword/ UML-notation	Tillgänglig inom klassen?	Tillgänglig inom samma paket?	Tillgänglig i subklass?	Tillgänglig för alla andra klasser?
private -	Ja	Nej	Nej	Nej
“package” ~	Ja	Ja	Nej	Nej
protected #	Ja	Ja	Ja	Nej
public +	Ja	Ja	Ja	Ja

Vad ärver subklassen

- Alla datamedlemmar ärvs, *inklusive private* och static, men…
  - Datamedlemmar som är privata i superklassen är inte synliga i subklassen - men de finns i objekten
- Alla metoder ärvs, *utom privata metoder samt konstruktorer*
  - Subklassens konstruktor kan anropa basklassens, `super(…);`
    Anropet ska alltid ske på först raden i subklassens konstruktor

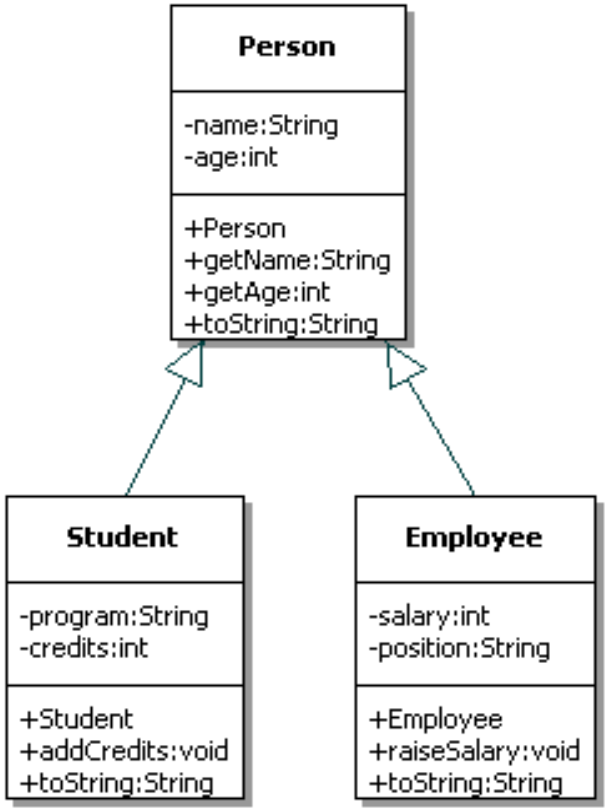
Omdefiniera en ärvd metod (override)

- Logiskt sett samma operation som i superklassen
- Metodens signaturer måste var identisk i subklassen: *returtyp*, namn, parameterlista

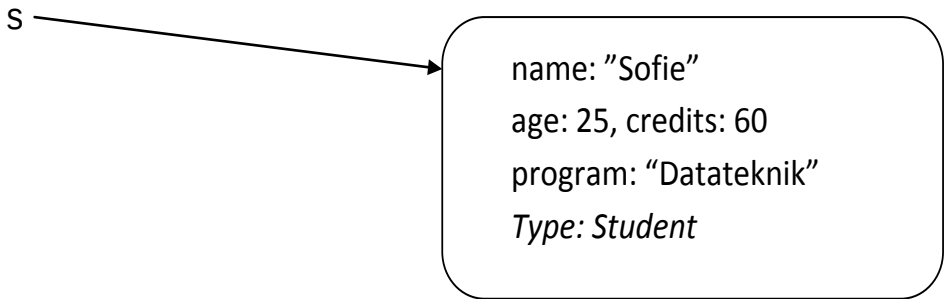
```
public class CreditAccount extends Account {
        . . .
        @Override
        public void withdraw(double amount) {
                if(balance+limit >= amount) { . . .
        }
        . . .
```

- Skilj omedfiniering av metod vid arv, override, från överlagring av metoder inom en klass, overload

Privat data vid arv




```
Student s =
new Student("Sofie",25,"Datateknik",60);
```



Privat data vid arv

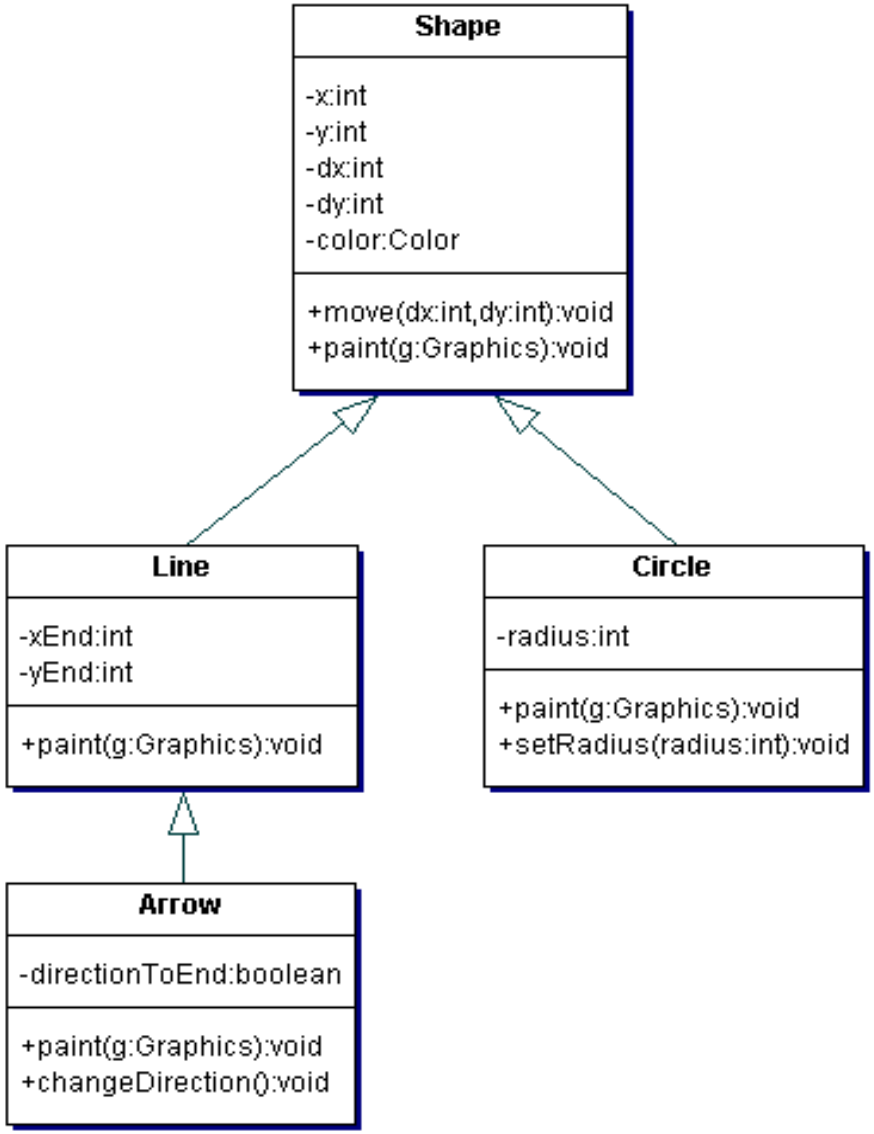
- Ärvda datamedlemmar med synligheten private kan ej refereras direkt i koden för subklassen!

```
public class Student extends Person {
     . . .
     public String toString() {

        String info = super.getName()
          + ", program  " + program + ", age  "
          + super.getAge() + ", " + credits +
          " credits";


         return info;
   }
```

- I detta fall går även detta (samma metod, getName, i superklass som i subklass)

```
   public String toString() {
      String info = getName()
```

Specialisering vs generalisering



Java och superklassen Object

- Alla klasser i Java ärver, direkt eller indirekt, från superklassen ”Object” (sic)
- `public class A { …`
  är detsamma som
  `public class A extends Object { …`
- Några ärvda metoder:
  - `public String toString`
  - `public boolean equals(Object other)`
  - `public int hashCode()`
- Dessa metoder bör normalt omdefinieras, override, i din klass
- Markera med omdefinerad metod med ”@Override”

Canonical form of a Java class

A canonical form (following well-established rules) of a Java class should provide the following

- a no-argument constructor
- methods for testing for object equality: equals and hashCode
- a method, toString, for String representation
- a method, clone, for cloning objects, if necessary making a *deep* copy
- an implementation of compareTo from interface Comparable defining the natural ordering for objects of this type

Canonical form of a Java class

```
public class Person implements Cloneable {

        private String name;
        private int age;
        . . .
        @Override
        public boolean equals(Object other) {
                if(this == other) return true;

                if(other instanceof Person) {
                        Person otherPerson = (Person) other;
                        return this.age == otherPerson.age  &&
                        this.name.equals(otherPerson.name);
                }

                return false;
        }

        . . .
```

Canonical form of a Java class

```
public class Person implements Cloneable {
        . . .
        public Object clone() {
                try {
                        Person copy = (Person) super.clone();
                        // Shallow copy, ok for Strings and ints
                        return copy;

                } catch (CloneNotSupportedException e) {
                        // This should never happen...
                        throw new Error();
                }
        }
```