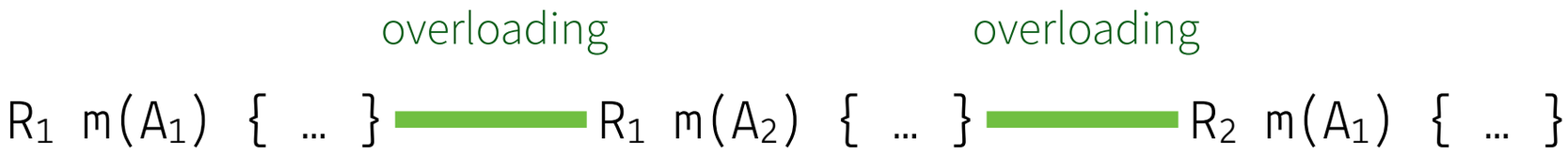
Föreläsning 20

Tobias Wrigstad

LSP, överlagring, overriding, subtypning, undantag, == och equals

Preamble

- Vad är en **typ**?
- Vad är en **subtyp**?
- Vad är relationen mellan **subklass** och **subtyp**?




```
C c = new C();
A2 a2 = new A2();
```

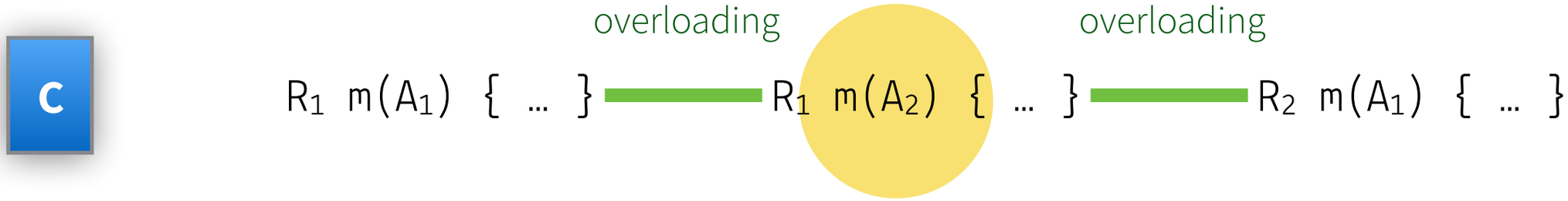
```
c.m(a2);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A2 a2 = new A2();
```

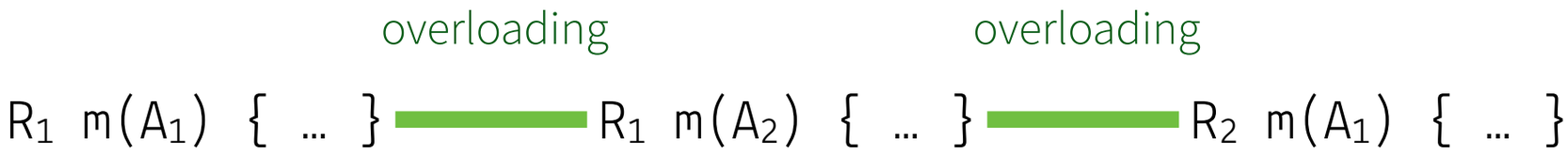
```
c.m(a2);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A1 a1 = new A1();
```

```
R2 r = c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel

overloading


```
C c = new C();
A1 a1 = new A1();
```

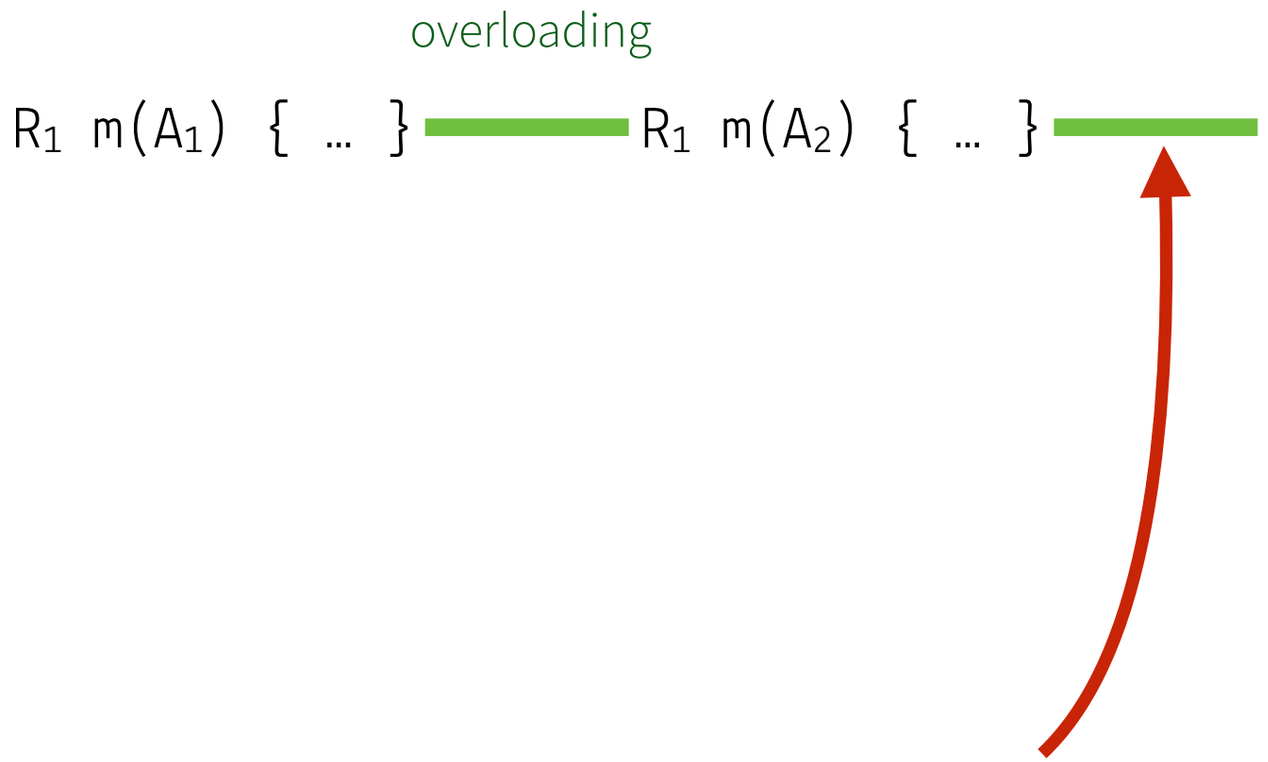
```
R2 r = c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A1 a1 = new A1();
```

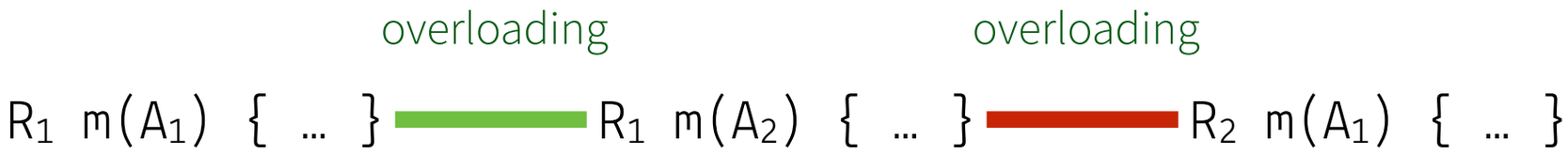
$$R_2\ r\ =\ c.m(a_1);$$

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A1 a1 = new A1();
```

```
R2 r = c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel

overloading


```
C c = new C();
A1 a1 = new A1();
```

```
R2 r = c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel

overloading

R_1 m(A_1) { … } —— R_1 m(A_2) { … }

```
C c = new C();
A1 a1 = new A1();
R1 r1 = new R1();
```

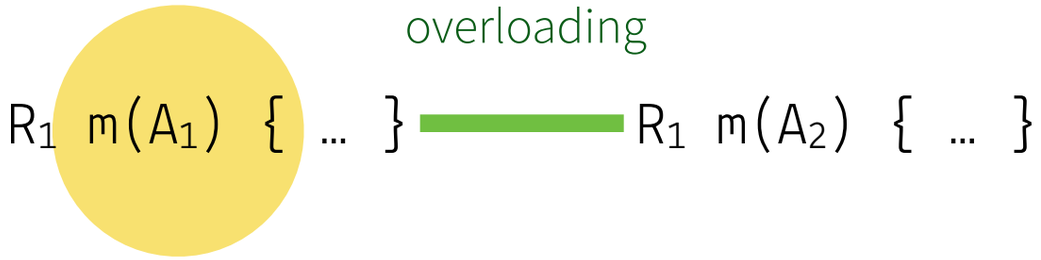
```
R1 r = c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A1 a1 = new A1();
R1 r1 = new R1();
```

$$R_1\ r\ =\ c.m(a_1);$$

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel

overloading

R_1 m(A_1) { … } —— R_1 m(A_2) { … }

```
C c = new C();
Object o = new A1();
```

```
c.m(o);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel

overloading

R_1 m(A_1) { … } —— R_1 m(A_2) { … }

```
C c = new C();
Object o = new A1();
```

```
c.m(o);
```

👎

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel

overloading

R_1 m(A_1) { … } —— R_1 m(A_2) { … }

```
C c = new C();
Object o = new A1();
```

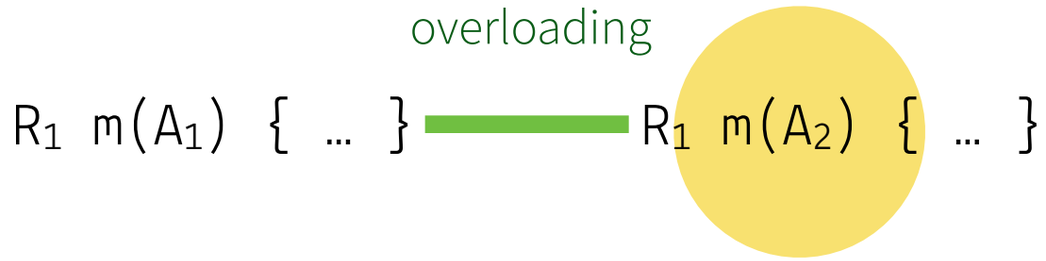
```
c.m((A2) o);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
Object o = new A1();
```

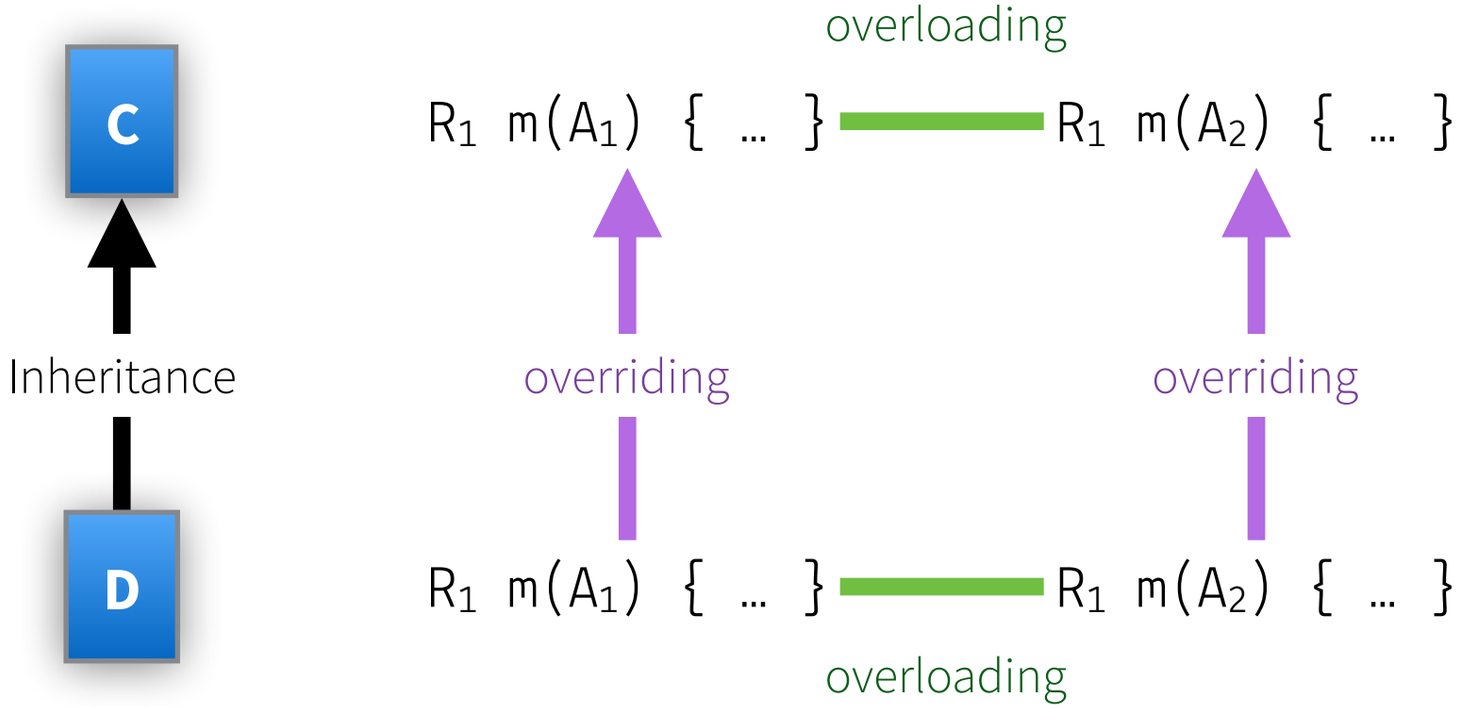
```
c.m((A2) o);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
D d = new D();
A1 a1 = new A1();
```

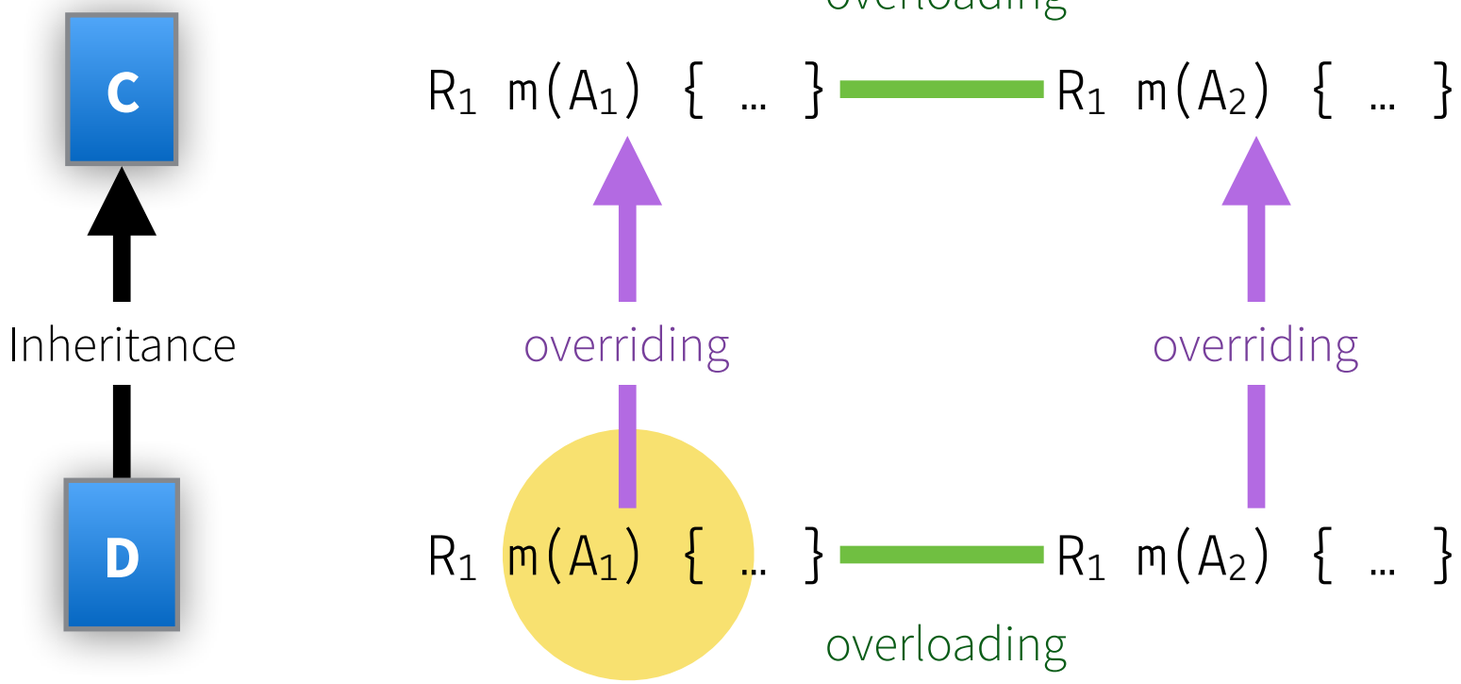
```
d.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
D d = new D();
A1 a1 = new A1();
```

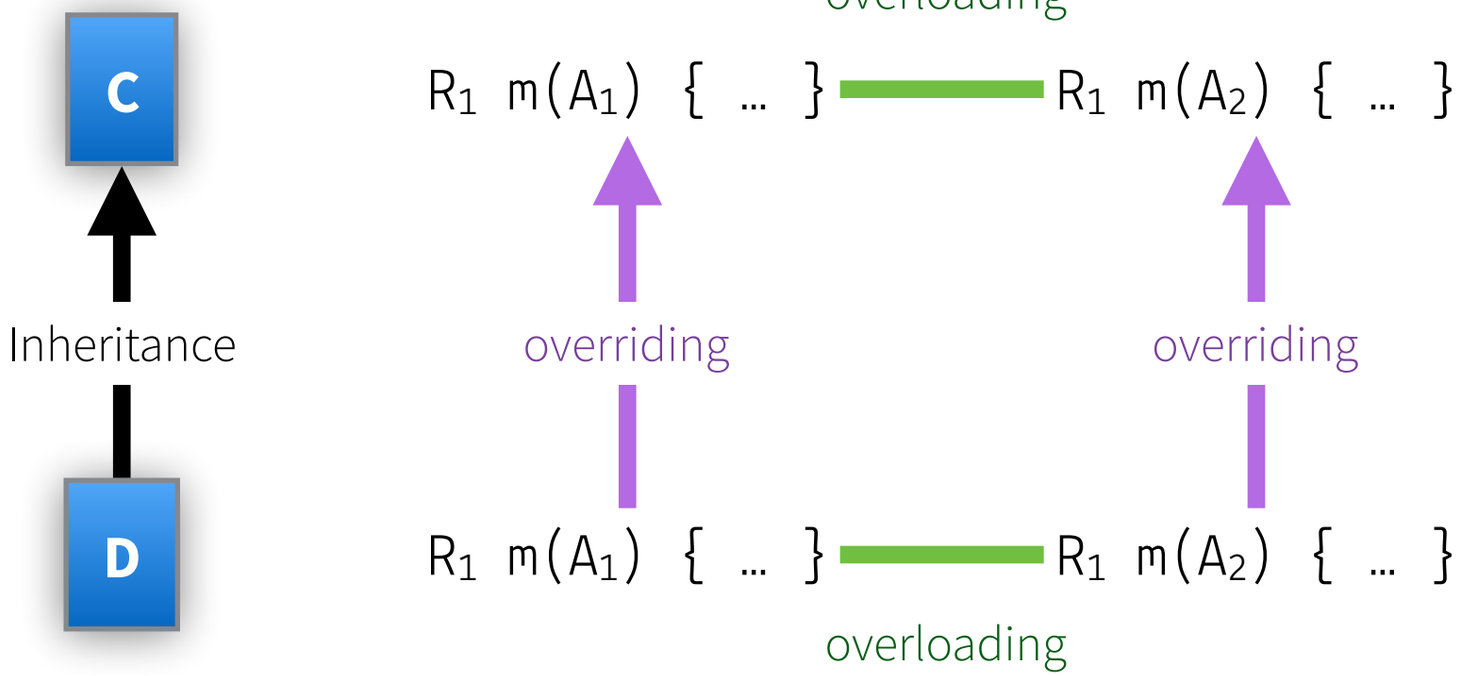
```
d.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A1 a1 = new A1();
```

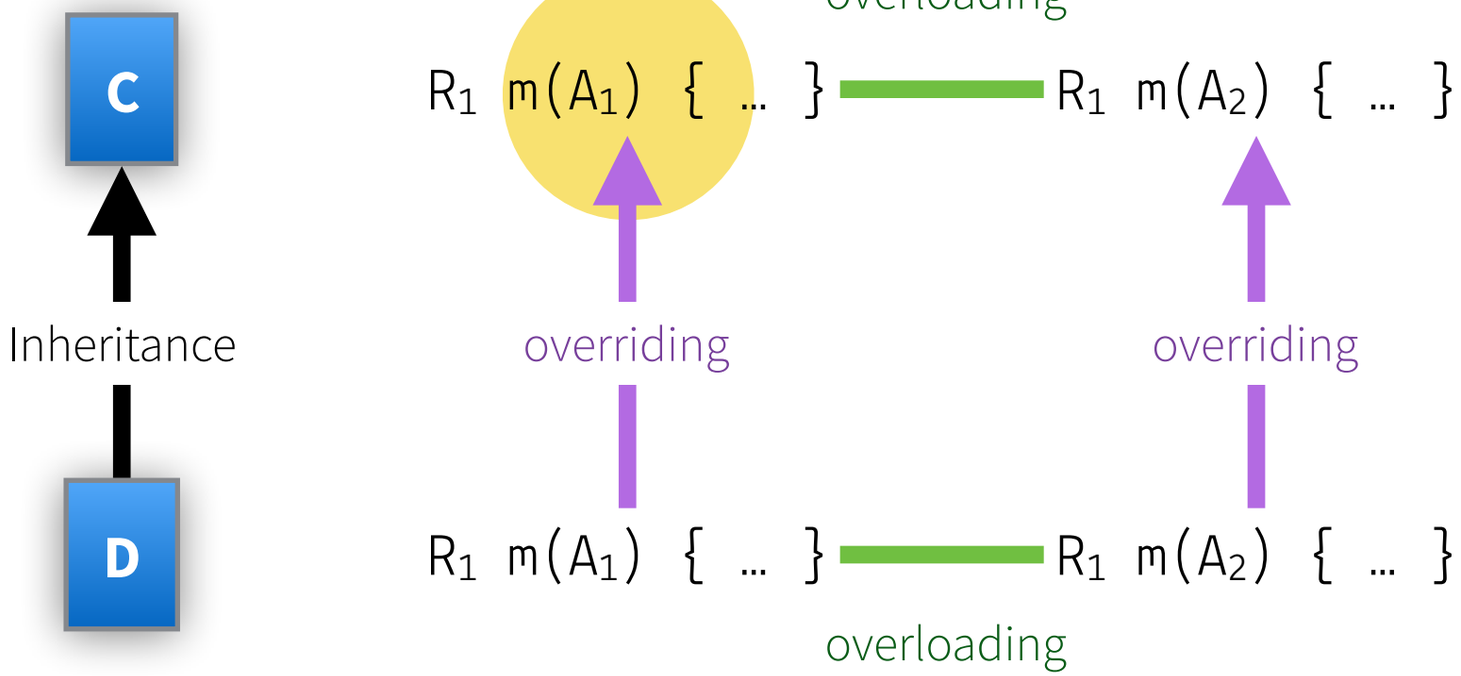
```
c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C c = new C();
A1 a1 = new A1();
```

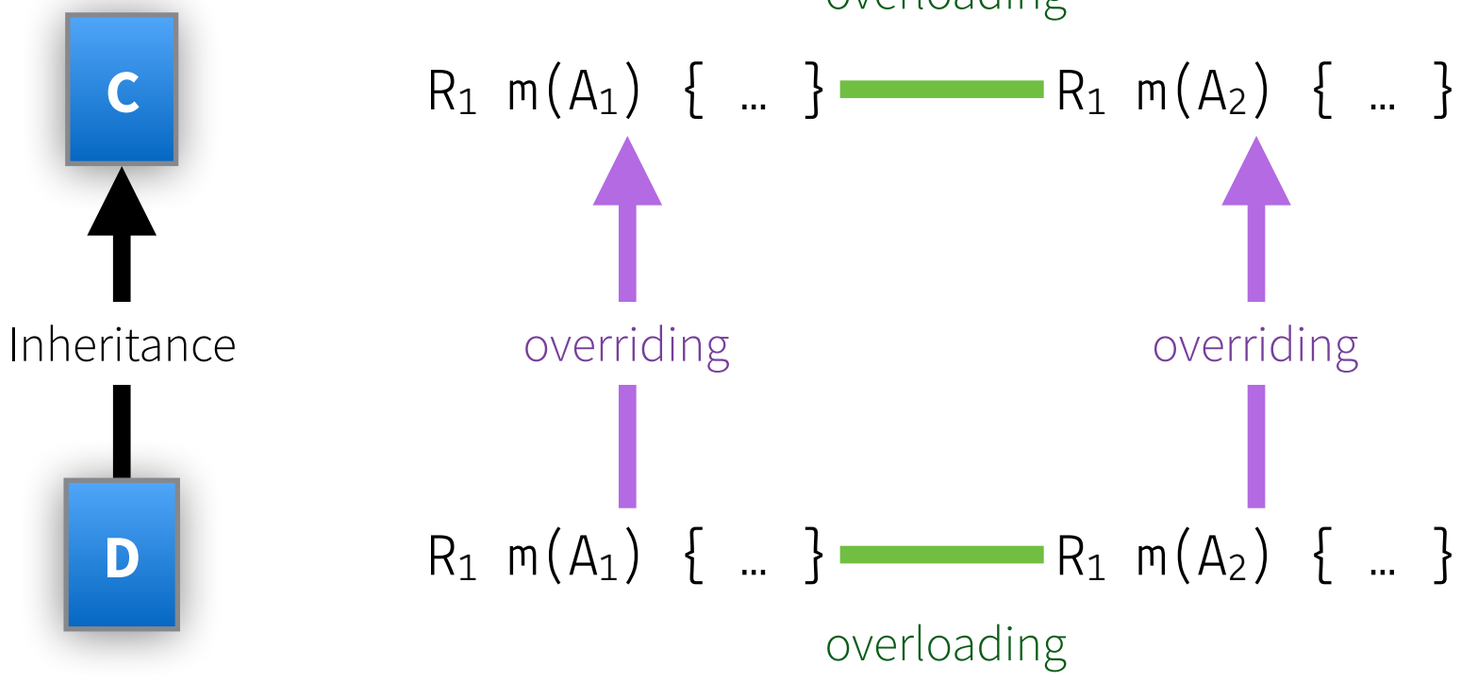
```
c.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

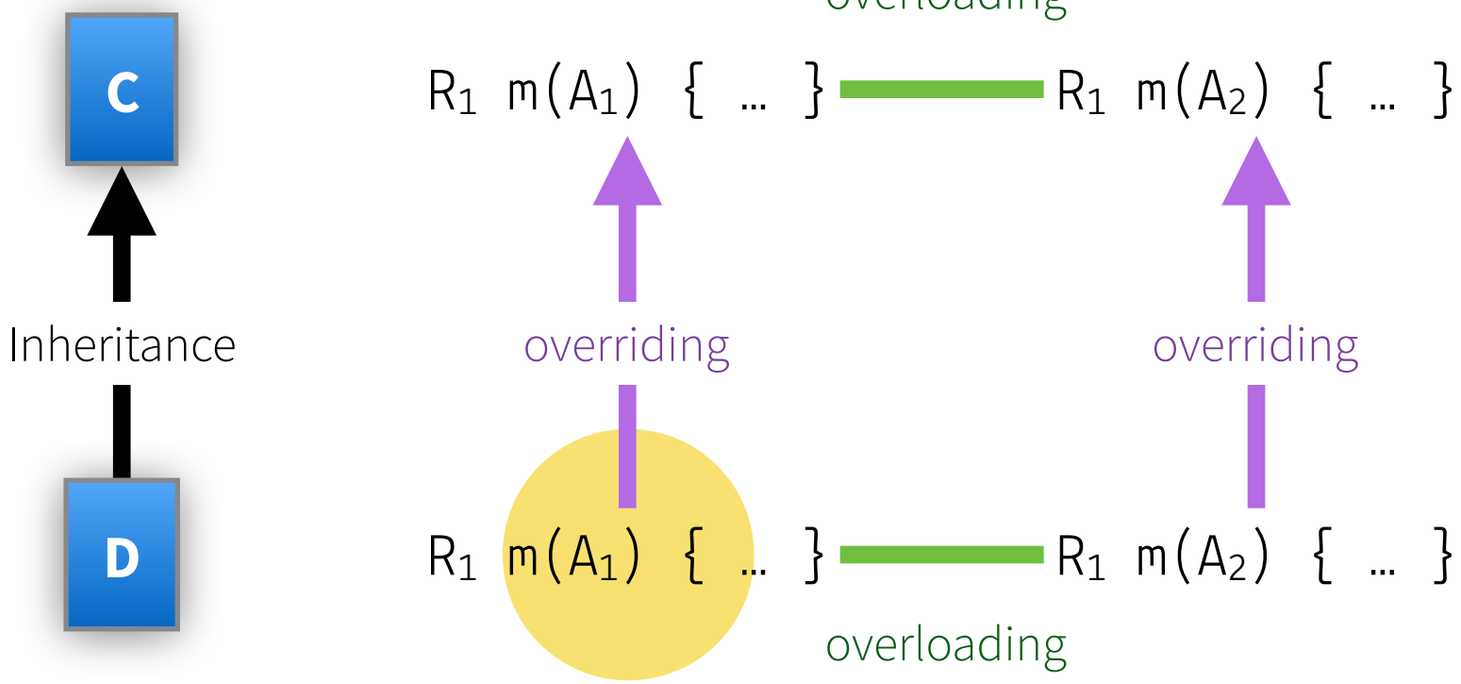
```
cd.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

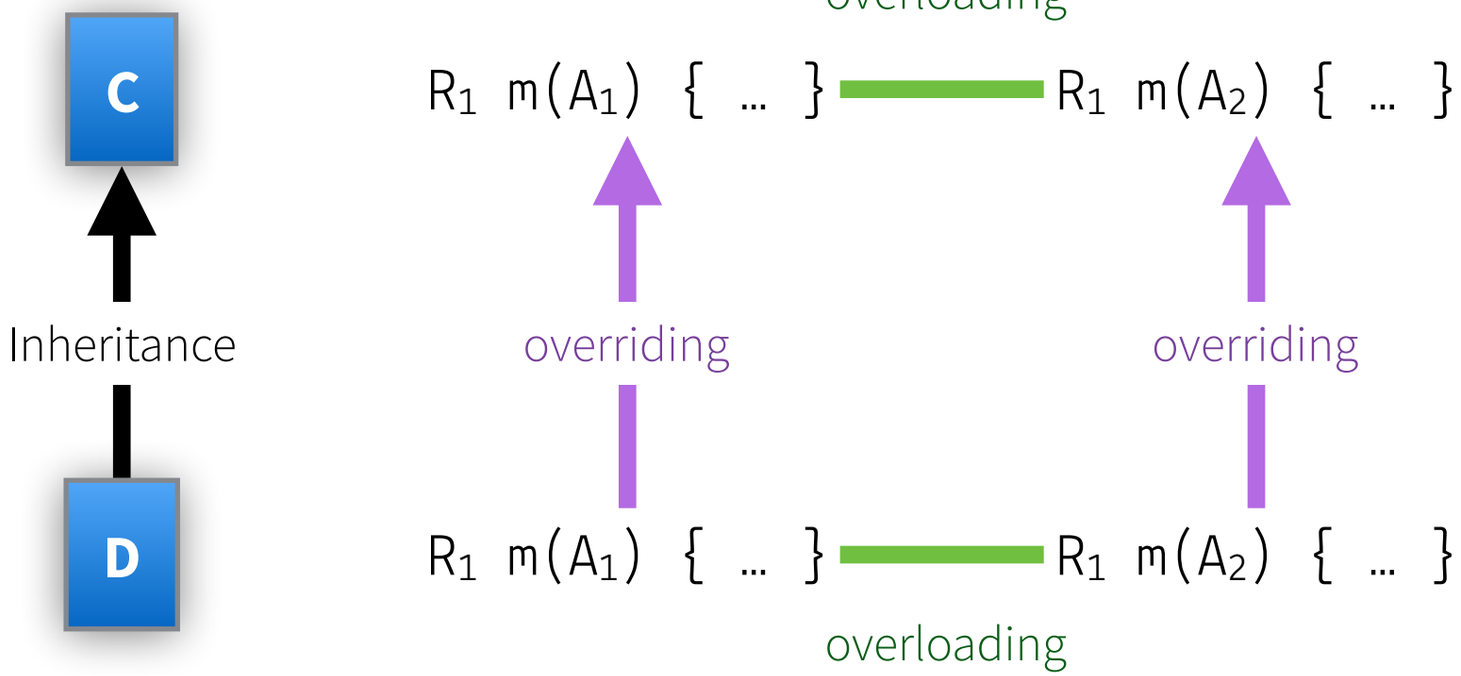
```
cd.m(a1);
```

A_1 = Instrument

A_2 = Bostad

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a = new A2();
```

OBS!

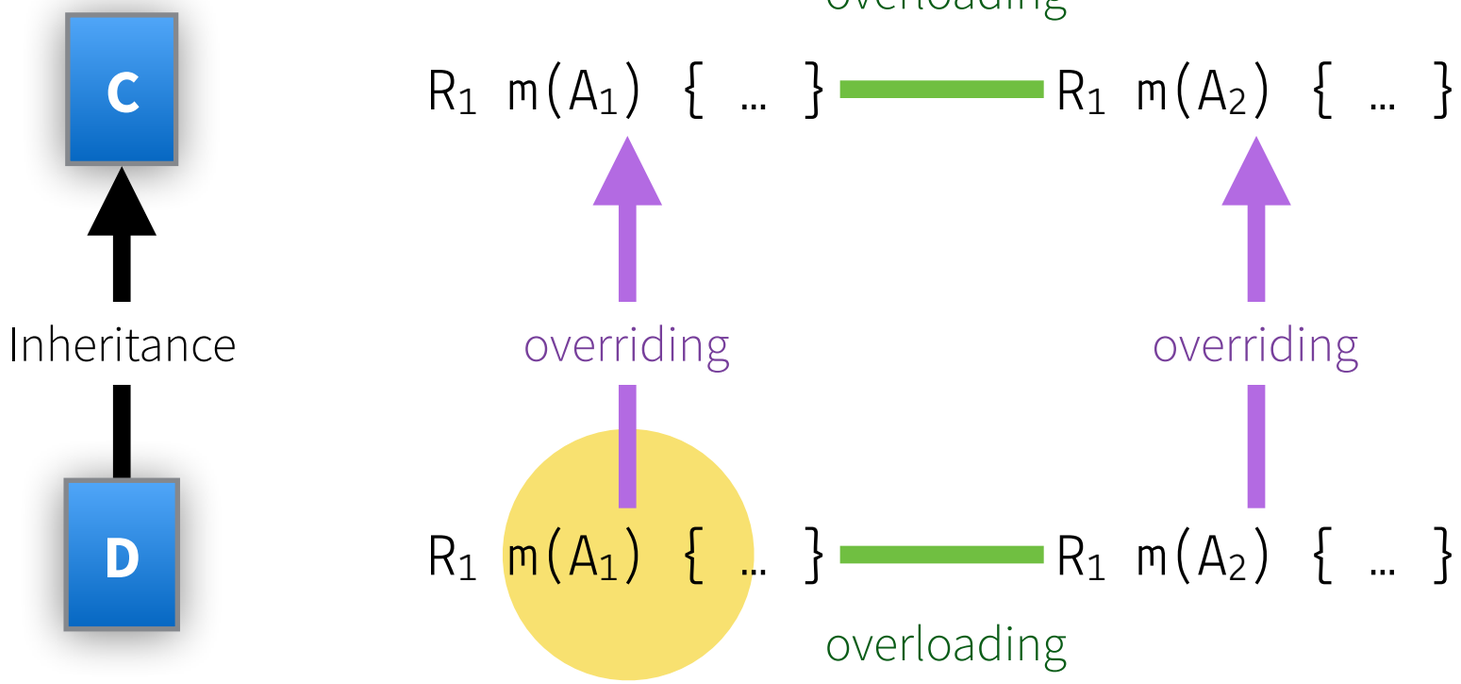
```
cd.m(a);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a = new A2();
```

OBS!

```
cd.m(a);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel

Sammanfattning

Overloading (överlagring) — flera definitioner av en funktion/metod/operator

Använd t.ex. antal argument och/eller argumentens typer för att välja funk/met/op

I Java tillåter vi överlagring baserat på antal argument och/eller argumentens typer

Sammanfattning

Overloading (överlagring) — flera definitioner av en funktion/metod/operator

Använd t.ex. antal argument och/eller argumentens typer för att välja funk/met/op

I Java tillåter vi överlagring baserat på antal argument och/eller argumentens typer

Dynamic dispatch

Vilken metod ett anrop binder till avgörs under körning (*aka dynamiskt*)

Sammanfattning

Overloading (överlagring) — flera definitioner av en funktion/metod/operator

Använd t.ex. antal argument och/eller argumentens typer för att välja funk/met/op

I Java tillåter vi överlagring baserat på antal argument och/eller argumentens typer

Dynamic dispatch

Vilken metod ett anrop binder till avgörs under körning (*aka dynamiskt*)

Single dispatch

Dynamiskt använder vi endast en typ för att avgöra vilken metod som körs

I Java är detta mottagarens (receiver) typ

Sammanfattning

Overloading (överlagring) — flera definitioner av en funktion/metod/operator

Använd t.ex. antal argument och/eller argumentens typer för att välja funk/met/op

I Java tillåter vi överlagring baserat på antal argument och/eller argumentens typer

Dynamic dispatch

Vilken metod ett anrop binder till avgörs under körning (*aka dynamiskt*)

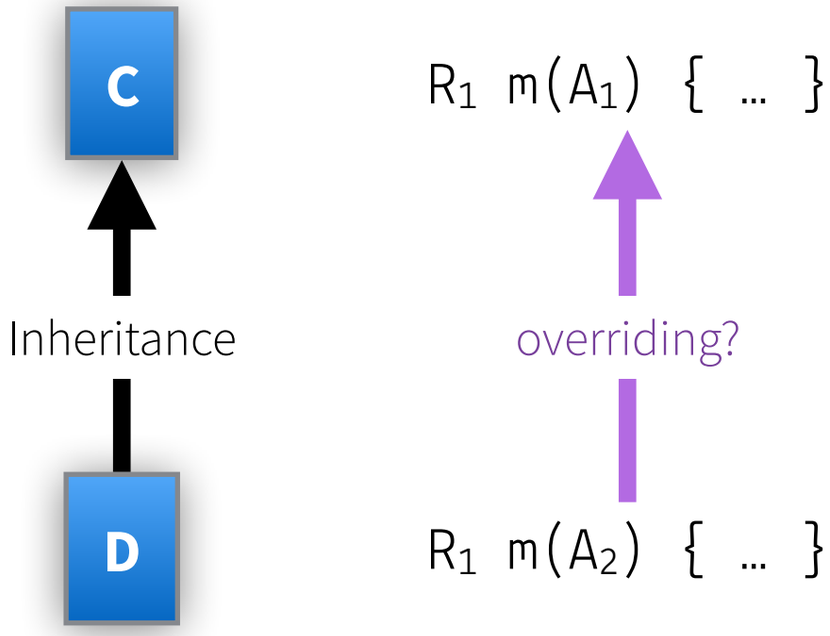
Single dispatch

Dynamiskt använder vi endast en typ för att avgöra vilken metod som körs

I Java är detta mottagarens (receiver) typ

Multiple dispatch

Existerar men är inte vanligt (se t.ex. CLOS, Julia)




```
C cd = new D();
A1 a1 = new A1();
```

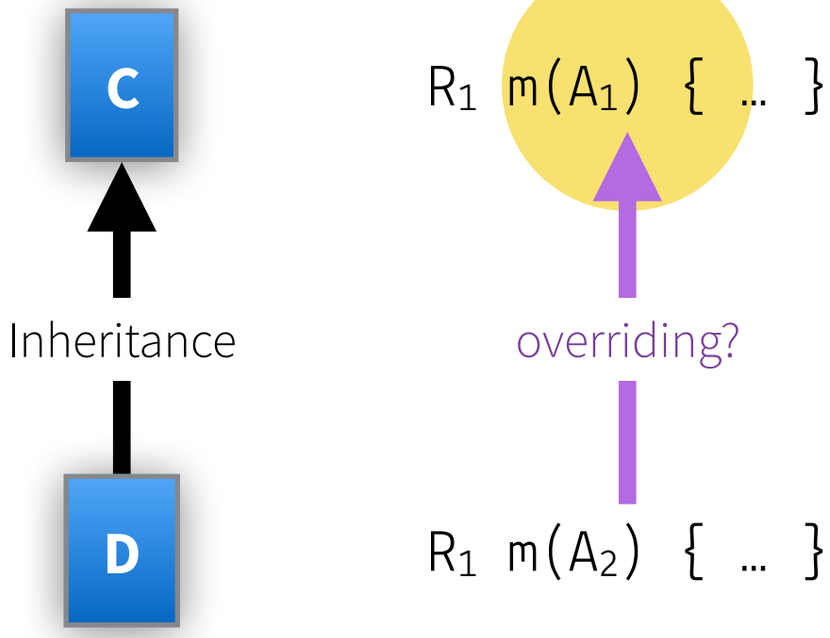
```
cd.m(a1);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

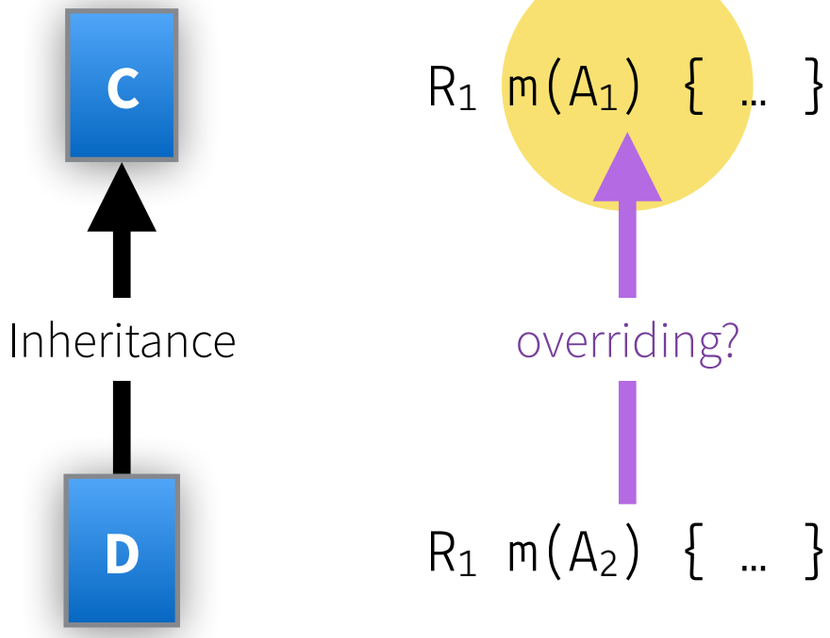
```
cd.m(a1);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel



Strengthening

```
C cd = new D();
A1 a1 = new A1();
```

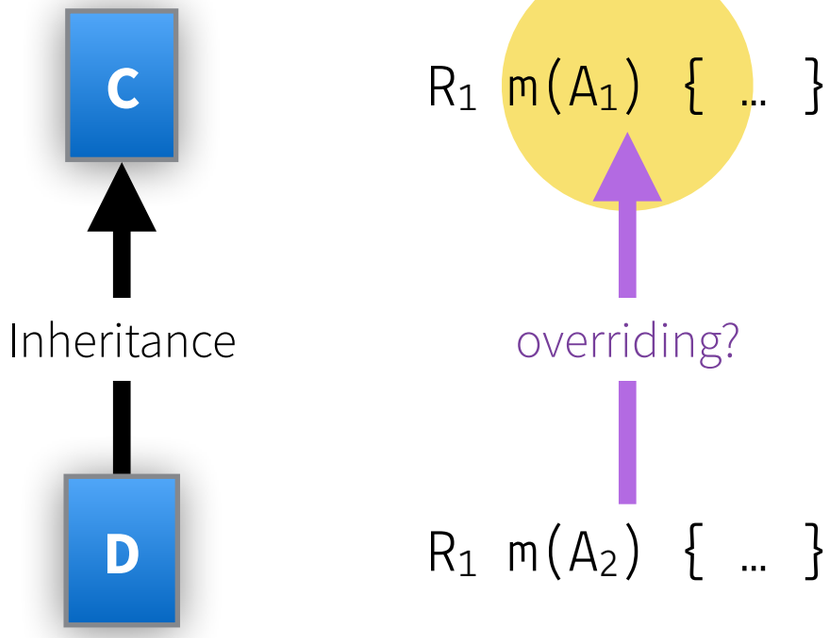
```
cd.m(a1);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel



Strengthening

```
C cd = new D();
A1 a1 = new A1();
```

```
cd.m(a1);
```

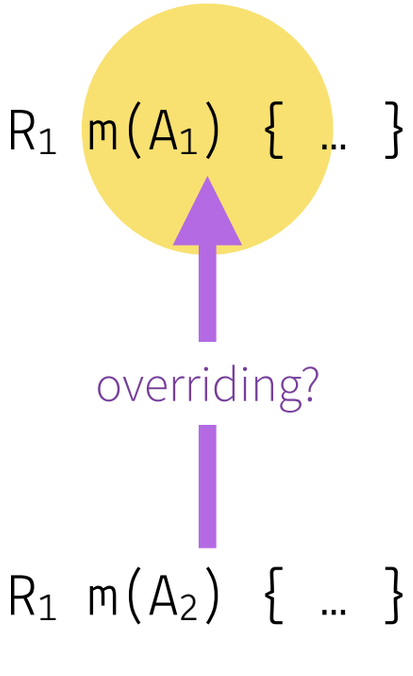
👎

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel



Strengthening

Det här är inte overriding utan overloading!

```
C cd = new D();
A1 a1 = new A1();
```

```
cd.m(a1);
```

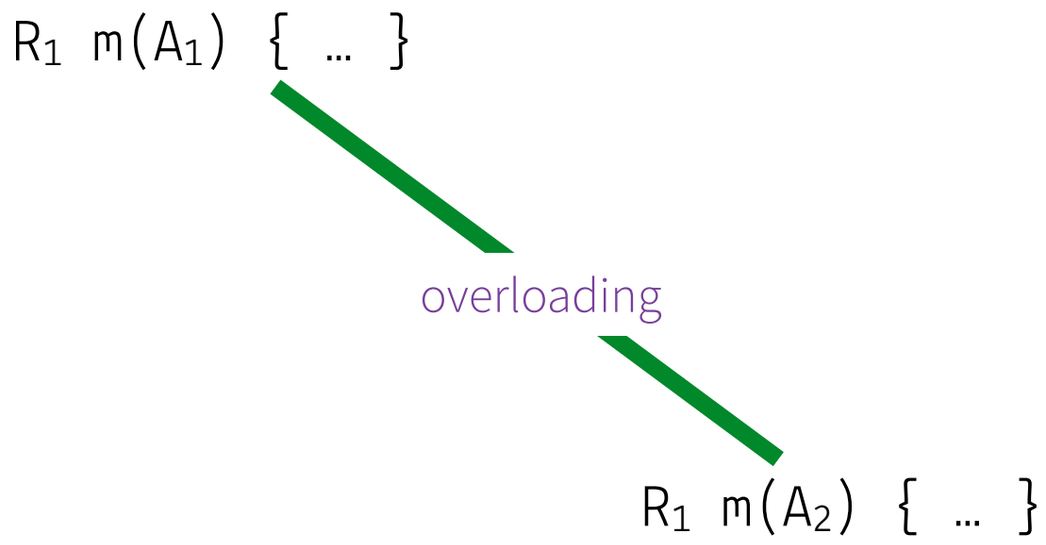
👎

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

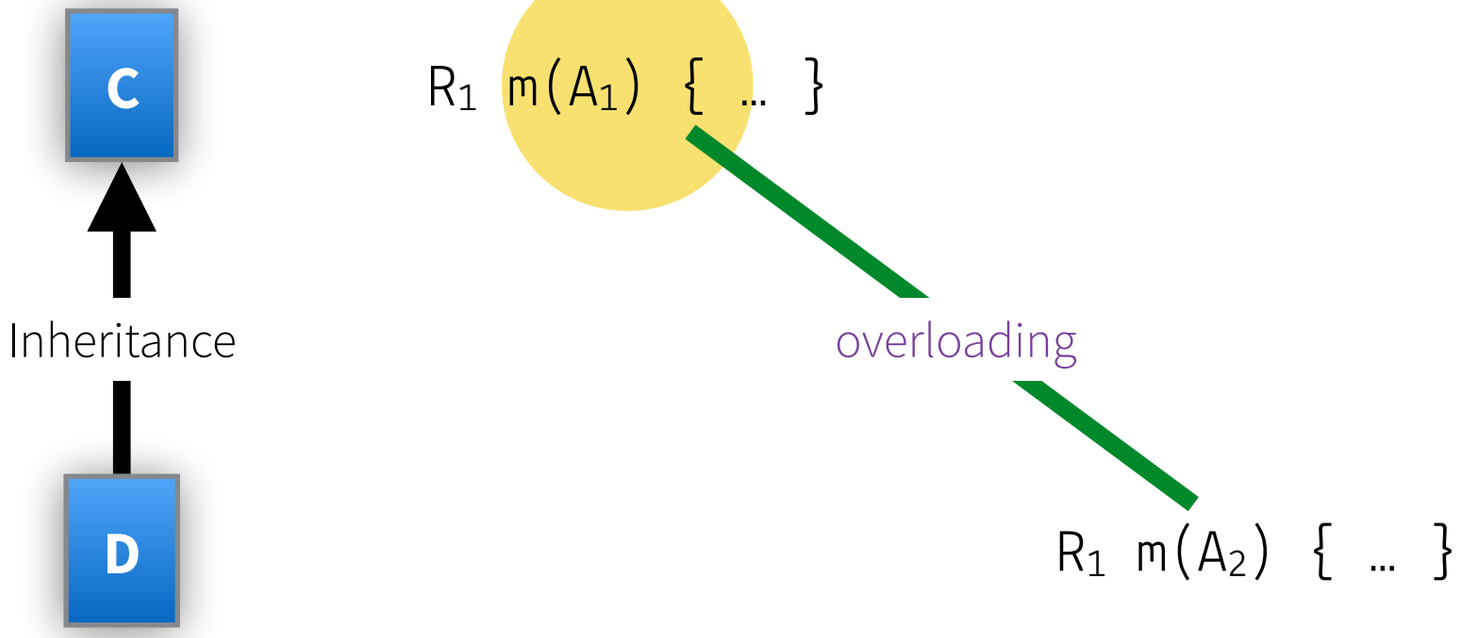
```
cd.m(a1);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

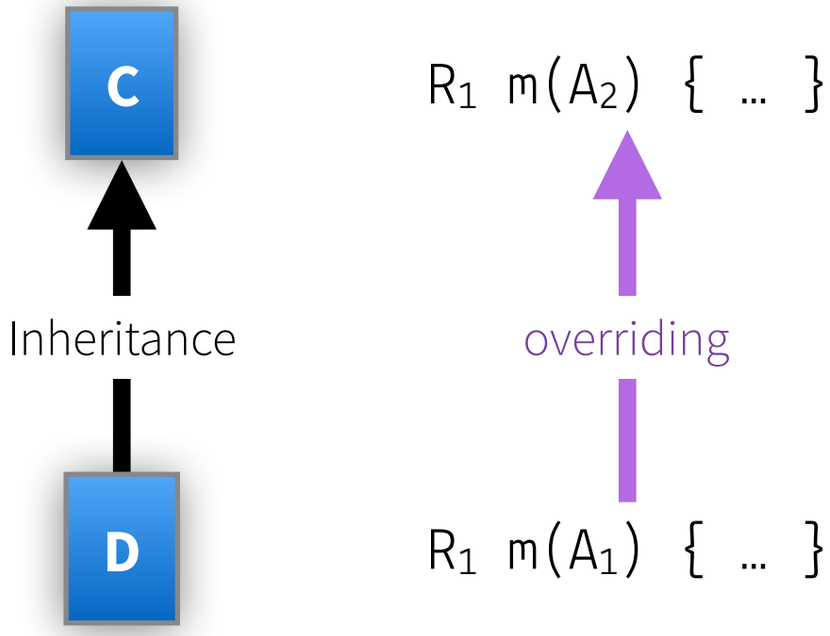
```
cd.m(a1);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel




```
C cd = new D();
A2 a2 = new A2();
```

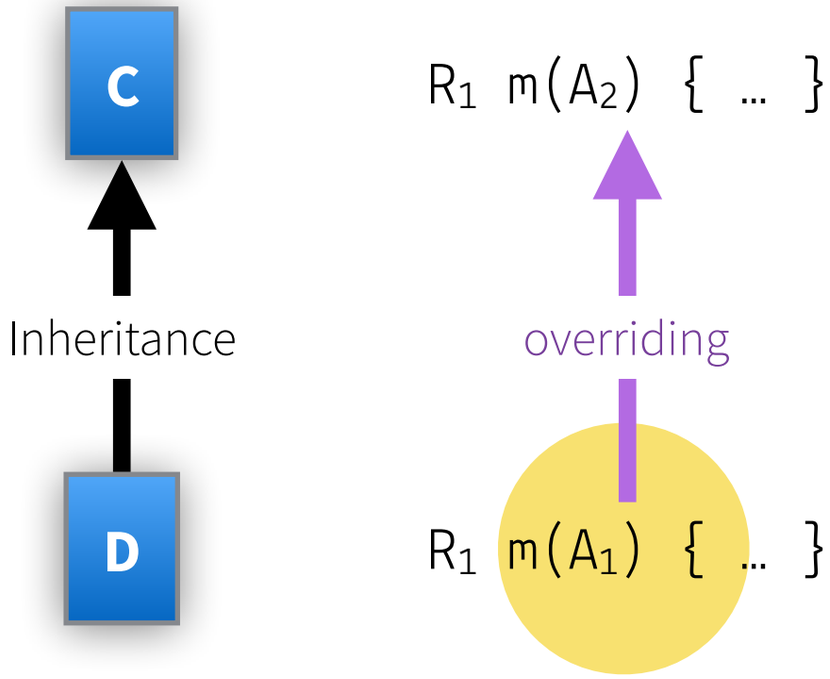
```
cd.m(a2);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel




```
C cd = new D();
A2 a2 = new A2();
```

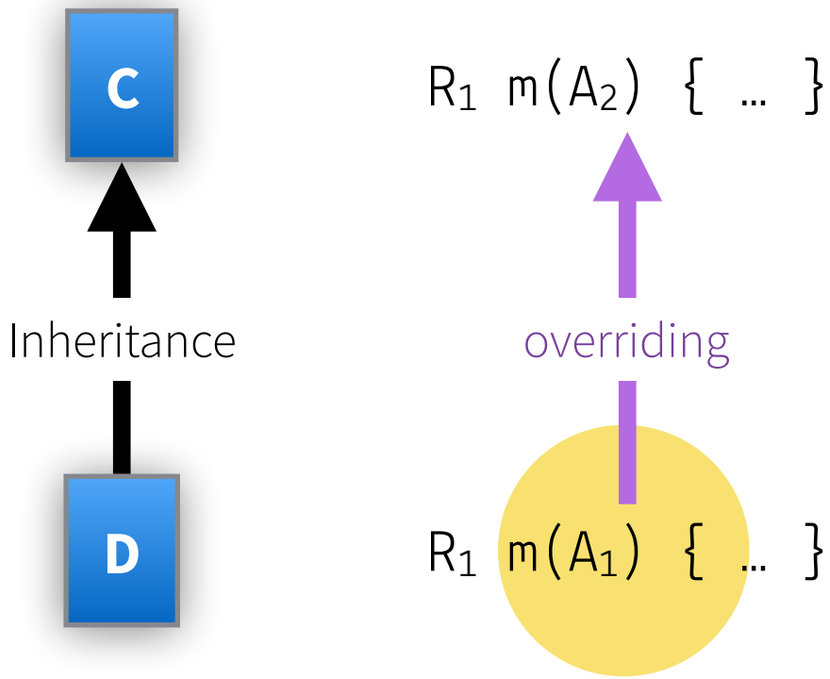
```
cd.m(a2);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel



Weakening

```
C cd = new D();
A2 a2 = new A2();
```

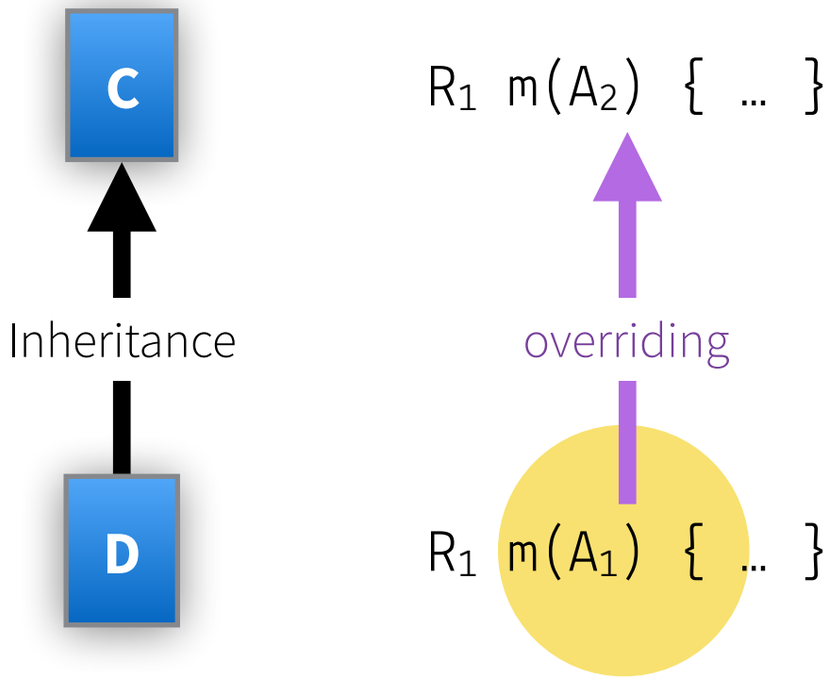
```
cd.m(a2);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel



Weakening

```
C cd = new D();
A2 a2 = new A2();
```

```
cd.m(a2);
```

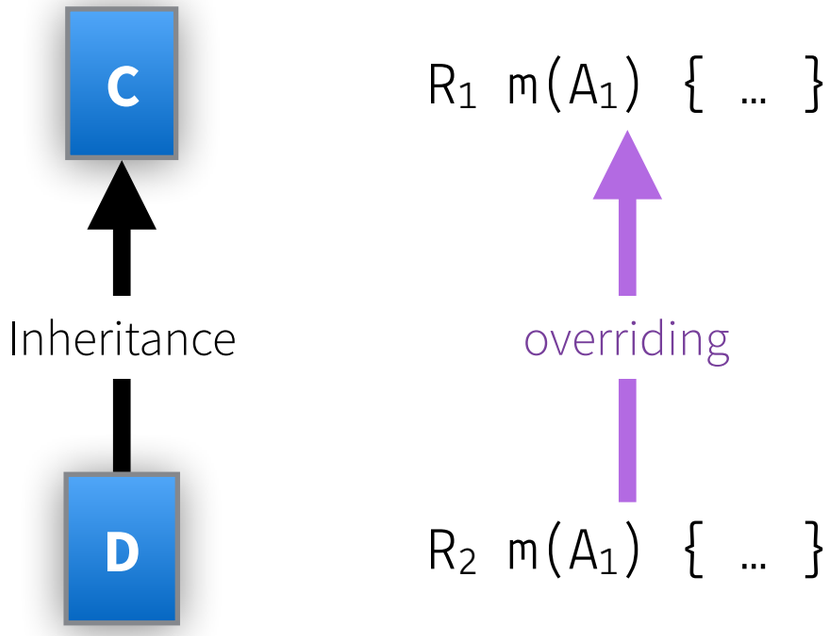
👍

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

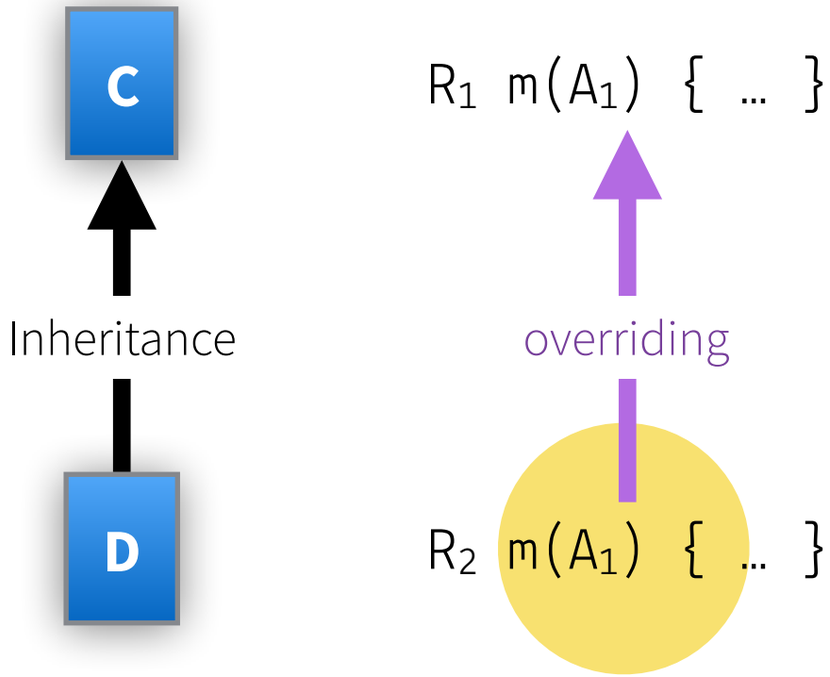
```
R1 r = cd.m(a1);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

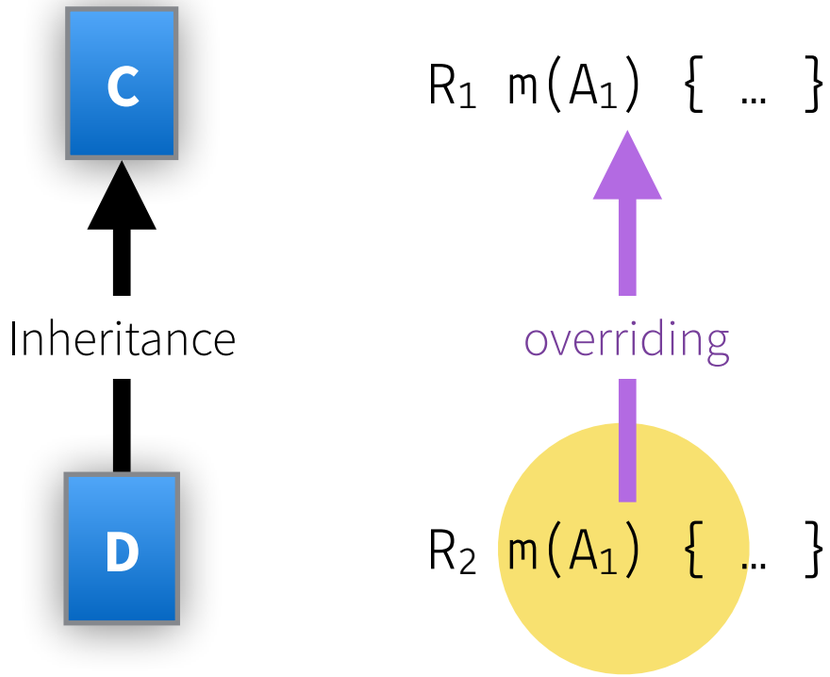
```
R1 r = cd.m(a1);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel



Strengthening

```
C cd = new D();
A1 a1 = new A1();
```

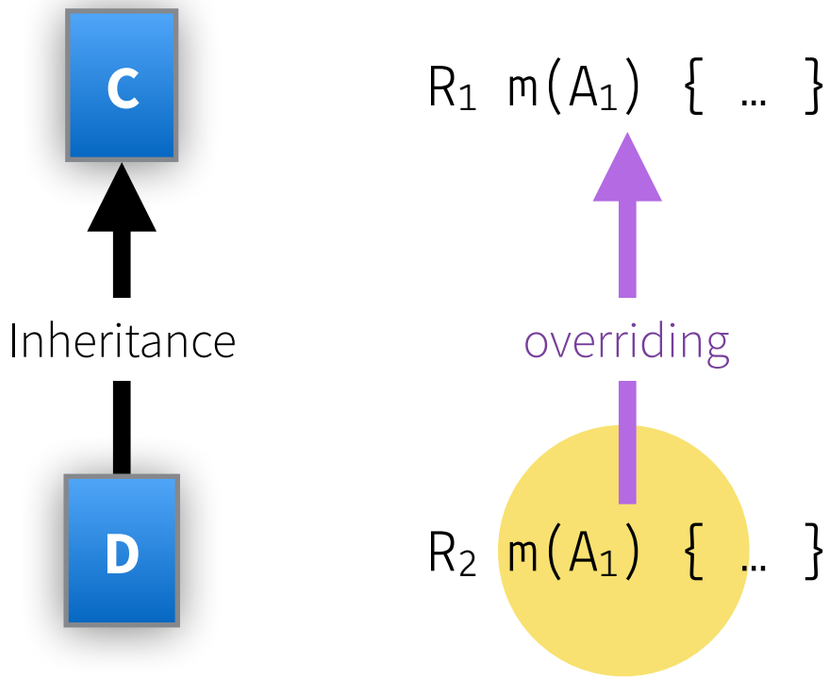
```
R1 r = cd.m(a1);
```

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel



Strengthening

```
C cd = new D();
A1 a1 = new A1();
```

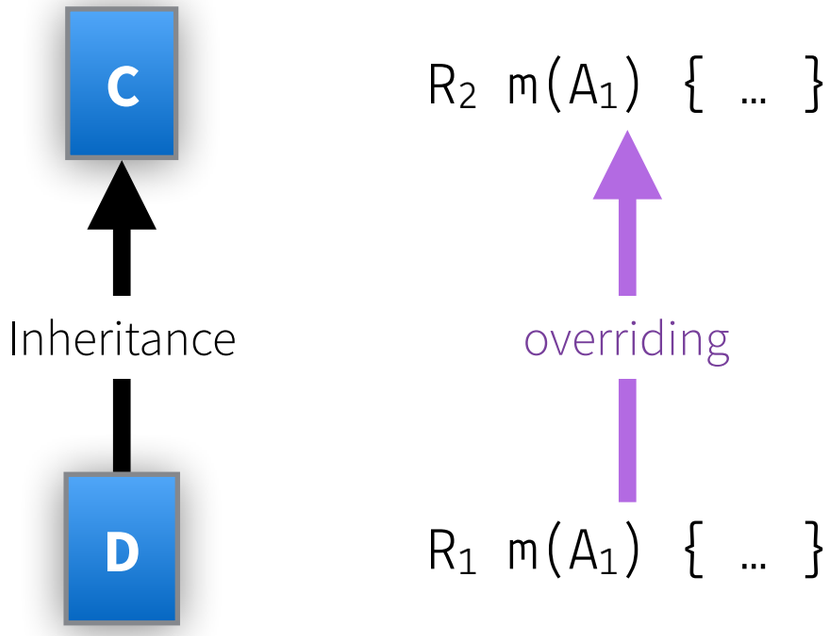
$R_1\ r = cd.m(a_1);$ 👍

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

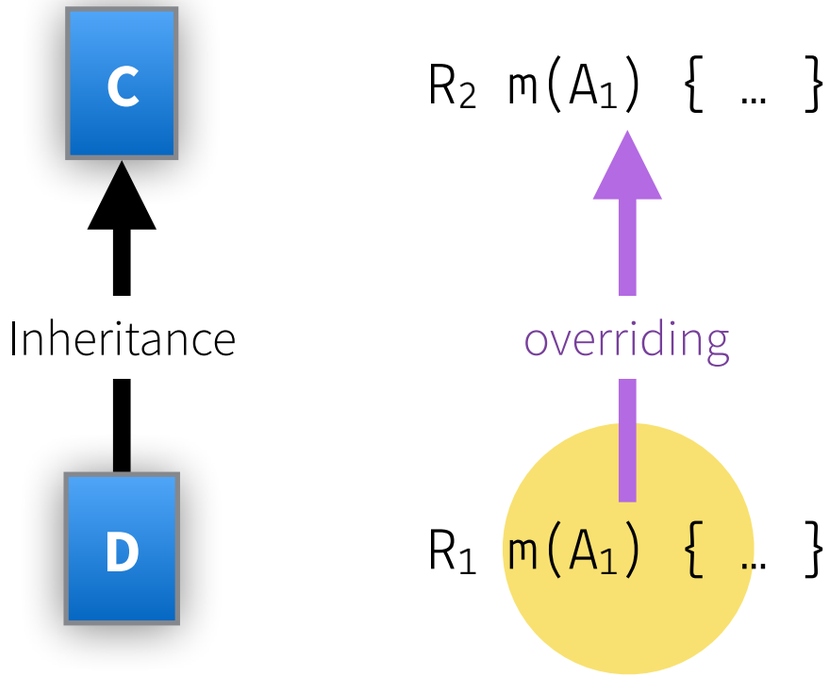
```
R2 r = cd.m(a1);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel




```
C cd = new D();
A1 a1 = new A1();
```

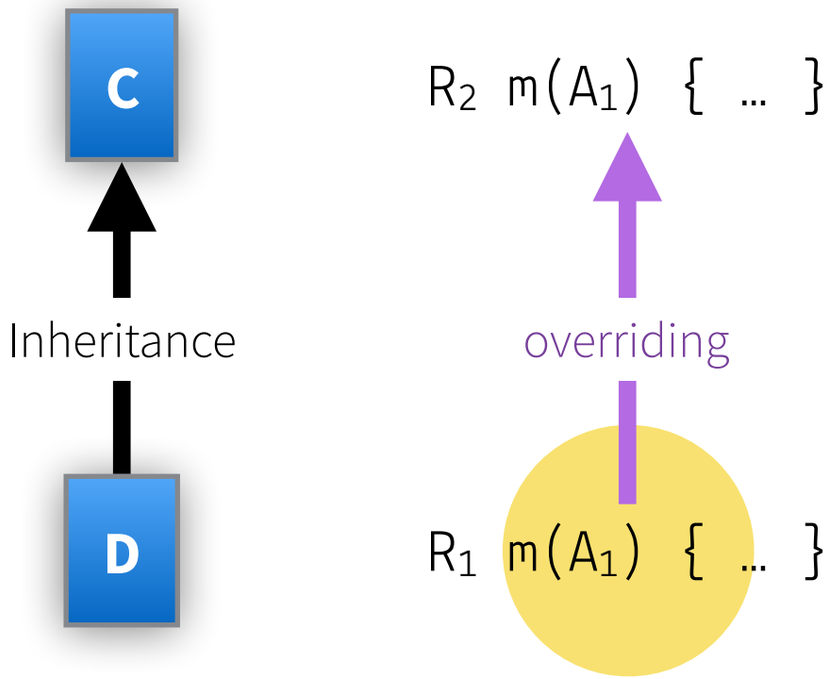
```
R2 r = cd.m(a1);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel



Weakening

```
C cd = new D();
A1 a1 = new A1();
```

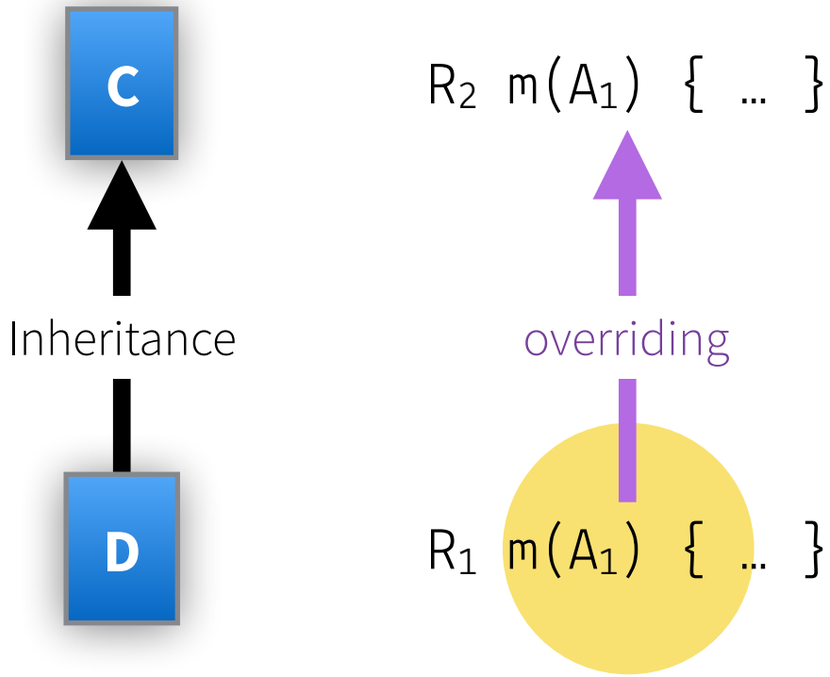
```
R2 r = cd.m(a1);
```

A1 = Instrument

A2 = Fiol

R1 = Geometrisk form

R2 = Rektangel



Weakening

```
C cd = new D();
A₁ a₁ = new A₁();
```

```
R₂ r = cd.m(a₁);
```

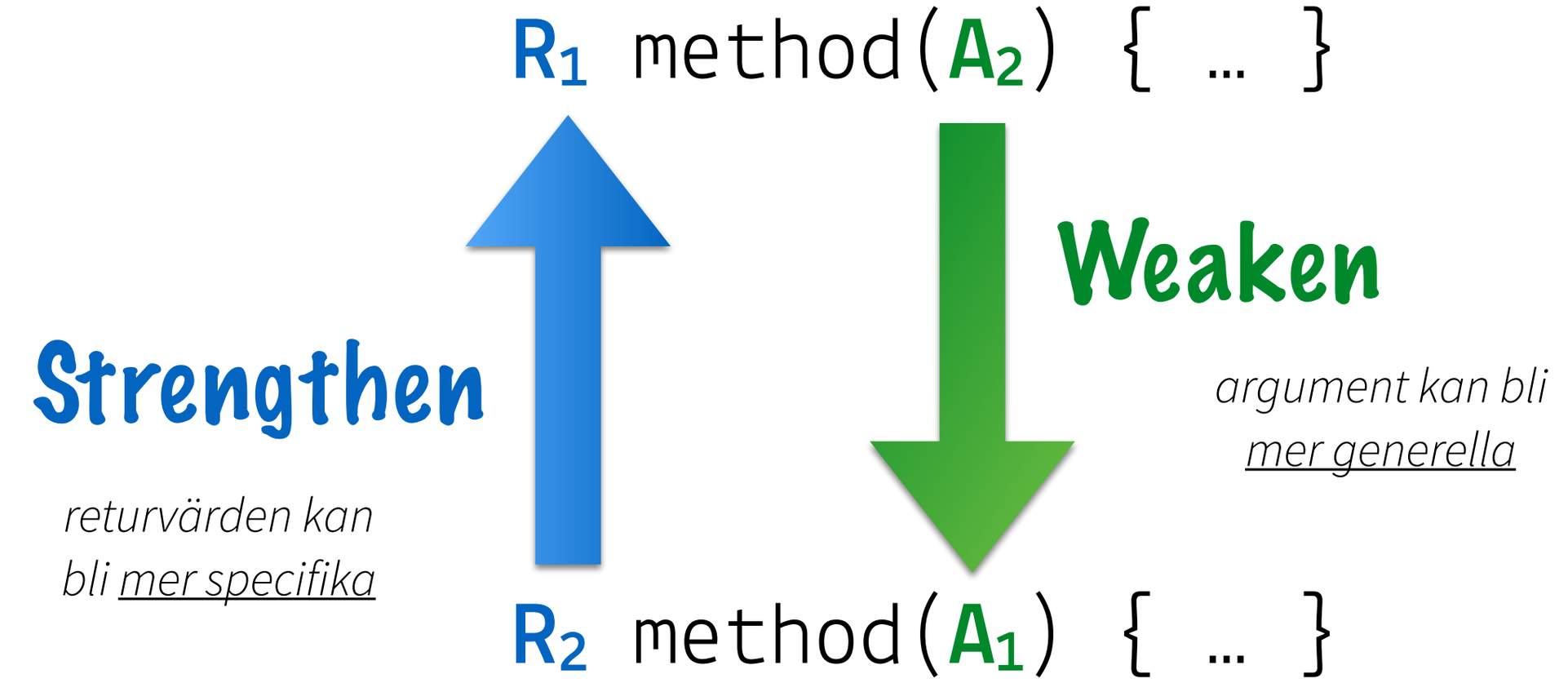
👎

A_1 = Instrument

A_2 = Fiol

R_1 = Geometrisk form

R_2 = Rektangel


R1 method(A2) { … }
Strengthen
returvärden kan bli mer specifika
Weaken
argument kan bli mer generella
R2 method(A1) { … }

Sammanfattning

Covariance (specifik → specifik)

En mer *specifik* metod *kan* returnera mer *specifika returvärden*

Contravariance (specifik → generell)

En mer *specifik* metod *kan* tillåta mer *generella argument*

Liskovs Substitutionsprincip

*Låt P vara ett program som bl.a. har objekt av typen T. Om S är en **subtyp** av T kan vi **ersätta** T-objekt med S-objekt i P utan att de eftersträvansvärde egenskaperna i programmet ändras.*

Barbara Liskov

ACM Turing Award

Liskovs Substitutionsprincip

*Låt P vara ett program som bl.a. har objekt av typen T. Om S är en **subtyp** av T kan vi **ersätta** T-objekt med S-objekt i P utan att de eftersträvansvärde egenskaperna i programmet ändras.*

Barbara Liskov

ACM Turing Award

- Contravariance för argument

Liskovs Substitutionsprincip

*Låt P vara ett program som bl.a. har objekt av typen T. Om S är en **subtyp** av T kan vi **ersätta** T-objekt med S-objekt i P utan att de eftersträvansvärde egenskaperna i programmet ändras.*

Barbara Liskov

ACM Turing Award

- Contravariance för argument
- Covariance för returtyper och exceptions

Liskovs Substitutionsprincip

*Låt P vara ett program som bl.a. har objekt av typen T. Om S är en **subtyp** av T kan vi **ersätta** T-objekt med S-objekt i P utan att de eftersträvansvärde egenskaperna i programmet ändras.*

Barbara Liskov
ACM Turing Award

- Contravariance för argument
- Covariance för returtyper och exceptions
- Behavioural constraints:

  Previllkor kan inte förstärkas

  Postvillkor kan inte försvagas

  Subtyper måste bevara supertypens samtliga invarianter

Postamble

- Vad är en **typ**?
- Vad är en **subtyp**?
- Vad är relationen mellan **subklass** och **subtyp**?

  Subklassning i de flesta språk ger inte automatiskt subtypning (á la Liskov)

Undantagshantering

- I Java hanteras fel via exceptions
- Man kan själv ”kasta (throw) ett undantag (exception)”

```
throw new Exception()
```

- Flyttar kontrollflödet till **närmaste matchande omslutande** **`catch`**-block
- Exempel på hur exceptions kan fångas:

```
try {
  Rectangle r = (Rectangle) someObject;
  int x = y / z;
} catch(ClassCastException e) {
   …
} catch(ArithmeticException e) {
   …
}
```


matchningsordning

Finally

```
void postMessage(User u, Server s, Message m) {
  try {
    Session session = s.logIn(u.id(), u.password());
    session.post(m);
  } catch (MalformedMessageException e) {
    u.notify(...);
  } finally {
    session.logout();
  }
}
```

- Körs alltid, oavsett utgång i **`try`**-blocket
- Tillåter oss att lämna tillbaka resurser (”städa”) oavsett vad som händer

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}

public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

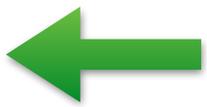
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

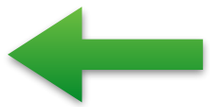
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException
is not a subclass of
IllegalArgumentException

B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException
is not a subclass of
IllegalArgumentException

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

`ArithmeticException` is not a subclass of `IllegalArgumentException`

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException is not a subclass of IllegalArgumentException

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException is not a subclass of IllegalArgumentException

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}

public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

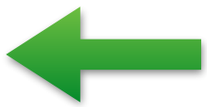
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

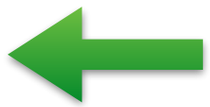
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

`ArithmeticException` is not a subclass of `IllegalArgumentException`

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

`ArithmeticException` is not a subclass of `IllegalArgumentException`

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException

ArithmeticException is a subclass of Exception

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException

ArithmeticException is a subclass of Exception

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException

ArithmeticException is a subclass of Exception

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (Exception e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException

ArithmeticException is a subclass of Exception

B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}

public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

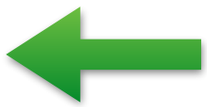
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

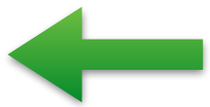
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException is a subclass of Exception

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

`ArithmeticException` is a subclass of `Exception`

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

ArithmeticException is a subclass of Exception

A
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

Exception!

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

`ArithmeticException` is a subclass of `Exception`

A
C

```java
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

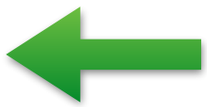
```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

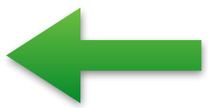
```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```



```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

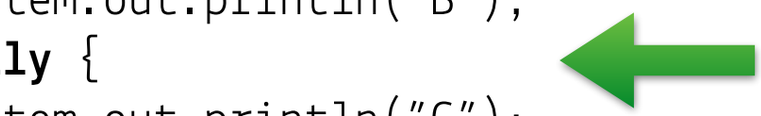
C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```



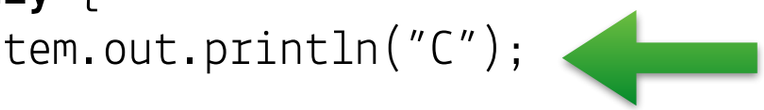
C

```
int foo(int a, int b) {
    return a + b;
}

int bar(int a) {
    return foo(a, 0);
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (Exception e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```



C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}

public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

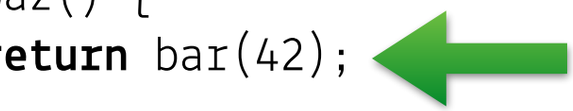
```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```



```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```java
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

```java
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

```
int foo(int a, int b) {
    return a / b;
}

int bar(int a) {
    try { return foo(a, 0); } finally { System.out.println("Z"); }
}

int baz() {
    return bar(42);
}
```

Exception!

```
public static void main(String[] args) {
    try {
        baz();
    } catch (IllegalArgumentException e) {
        System.out.println("A");
    } catch (ArithmeticException e) {
        System.out.println("B");
    } finally {
        System.out.println("C");
    }
}
```

Z
B
C

Undantagsinformation propageras genom effekter

```
void postMessage(User u, Server s, Message m) {
  Session session = s.logIn(u.id(), u.password());
  session.post(m);
}

void post(Message m) throws MalformedMessageException
```

Undantagsinformation propageras genom effekter

```
void postMessage(User u, Server s, Message m) {
  Session session = s.logIn(u.id(), u.password());
  session.post(m);
}

void post(Message m) throws MalformedMessageException
```

- Om `MalformedMessageException` ärver av `Exception` är den ”**checked**”

  Kräver att postMessage också **throws** `MalformedMessageException`, alternativt har ett `catch`-block runt anropet

Undantagsinformation propageras genom effekter

```
void postMessage(User u, Server s, Message m) {
  Session session = s.logIn(u.id(), u.password());
  session.post(m);
}

void post(Message m) throws MalformedMessageException
```

- Om `MalformedMessageException` ärver av `Exception` är den ”**checked**”

  Kräver att postMessage också **throws** `MalformedMessageException`, alternativt har ett `catch`-block runt anropet

- Om `MalformedMessageException` ärver av `RuntimeException` är den ”**unchecked**”

  Behöver varken fångas eller explicit propageras

  jmf. `NullPointerException`

Om MalformedMsg är ett Checked Exception

```
void post(Message m) throws MalformedMsgException
```

propagera

```
void postMessage(User u, Server s, Message m) throws MalformedMsgException {
  Session session = s.logIn(u.id(), u.password());
  session.post(m);
}
```

eller

```
void postMessage(User u, Server s, Message m) {
  Session session = s.logIn(u.id(), u.password());
  try {
    session.post(m);
  } catch(MalformedMsgException e) {
    …
  }
}
```

hantera

Definiera egna exceptions

```
/// Checked
class MalformedMessageException extends Exception { ... }

/// Unchecked
class MalformedMessageException extends RuntimeException { ... }
```

Värdesemantik eller pekarsemantik?

```
class Pair { Object fst; Object snd; }

void foo(Pair o) {
    o.snd = 4711;
}
```

```
public static void main(String[] args) {
    Pair p = new Pair();
    p.foo = 42;
    foo(p);
    System.out.println(p.snd);
}
```

4711

or

null

Identiska par?

```
class Pair {
  Object fst; Object snd;
  public Pair(Object a, Object b) { fst = a; snd = b; }
}

public static void main(String[] args) {
    Pair p1 = new Pair(42, 4711);
    Pair p2 = new Pair(42, 4711);
    System.out.println(p1 == p2);
}
```

`true`

or

`false`

Ekvivalenta par?

```
class Pair {
  Object fst; Object snd;
  public Pair(Object a, Object b) { fst = a; snd = b; }
}

public static void main(String[] args) {
    Pair p1 = new Pair(42, 4711);
    Pair p2 = new Pair(42, 4711);
    System.out.println(p1.equals(p2));
}
```

`true`

or

`false`

Standarddefinitionen av equals i Object

```
public class Object {
    public boolean equals(Object o) {
        return this == o;
    }
}
```

Definitionen av ekvivalenta par

```
public class Pair {
    private Object fst;
    private Object snd;

    public boolean equals(Pair o) {
        return this.fst.equals(o.fst) &&
               this.snd.equals(o.snd);
    }
}
```

Definitionen av ekvivalenta par

```
public class Pair {
    private Object fst;
    private Object snd;

    public boolean equals(Pair o) {
        return this.fst.equals(o.fst) &&
               this.snd.equals(o.snd);
    }
}
```

Kontravarians för argument!!!

Klassiskt fel: överlagring, inte overriding av equals

```
public class Pair {
    private Object fst;
    private Object snd;

    public boolean equals(Pair o) {
        return this.fst.equals(o.fst) &&
               this.snd.equals(o.snd);
    }
}
```

```
Object p1 = new Pair(1, 2);
Object p2 = new Pair(1, 2);
System.out.println(p1.equals(p2));
```

false

Klassiskt fel: överlagring, inte overriding av equals

```
public class Pair {
    private Object fst;
    private Object snd;

    public boolean equals(Pair o) {
        return this.fst.equals(o.fst) &&
               this.snd.equals(o.snd);
    }
}
```

```
Object p1 = new Pair(1, 2);
Object p2 = new Pair(1, 2);
System.out.println(p1.equals(p2));
```

Anrop till `Object`:s `equals()`!

`false`

Definitionen av ekvivalenta par

```
public class Pair {
    private Object fst;
    private Object snd;

    public boolean equals(Object o) {
        if (o instanceof Pair) {
            return this.equals((Pair) o);
        } else {
            return super.equals(o);
        }
    }

    public boolean equals(Pair o) {
        return this.fst.equals(o.fst) &&
               this.snd.equals(o.snd);
    }
}
```

Definitionen av ordnade par

```
public class Pair<T> implements Comparable<Pair> {
    private T fst;
    private T snd;

    public int compareTo(Pair o) {
        ...
    }
}
```

Definitionen av ordnade par

```
public class Pair<T> i                    ble<Pair> {
    private T fst;
    private T snd;

    public int compareTo(Pair o) {
        ...
    }
}
```

Oklar typ T!

Vi vet inte nog för att jämföra innehållen!

Definitionen av ordnade par

```
public class Pair<T extends Comparable> implements Comparable<Pair> {
    private T fst;
    private T snd;

    public int compareTo(Pair o) {
        ... this.fst.compareTo(o.fst)
        }
}
```

Definitionen av ordnade par

```
public class Pair<T extends Comparable> implements Comparable<Pair> {
    private T fst;
    private T snd;

    public int compareTo(Pair o) {
        ... this.fst.compareTo(o.fst)
    }
}
```

Tillåter oss att jämföra vilka typer av par som helst!

Definitionen av ordnade par

```
public class Pair<T extends Comparable<T>>
                                    implements Comparable<Pair<T>> {
    private T fst;
    private T snd;

    public int compareTo(Pair<T> o) {
        ... this.fst.compareTo(o.fst)
    }
}
```

Sammanfattning

Subklasser ger inte subtypes automatiskt — vi måste aktivt arbeta för detta

- Liskovs substitutionsprincip guidar oss, och motiverar varför subtyper är A Good Thing™
- Experimentera med detta tills du förstår varför och hur!

Java använder undantagshantering för felhantering

- Finally används också för resurshantering orelaterat till undantag
- Utdelad kod denna föreläsning: Undantagshantering.java — experimentera!

Identitet och inkapsling är superviktiga begrepp

- Klassiskt fel: `public boolean equals(T o) { … }` där `T` ≠ `Object`
- Java-idiom: `a.equals(b)` ⇒ `a.hashCode() == b.hashCode()`