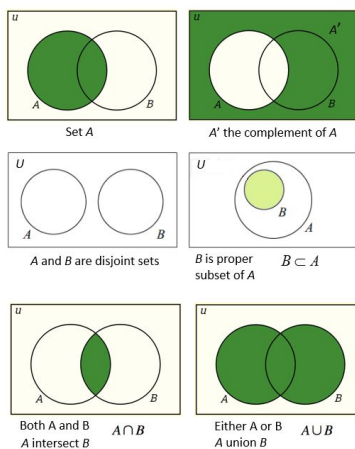


ex3supersub

Joachim von Hacht

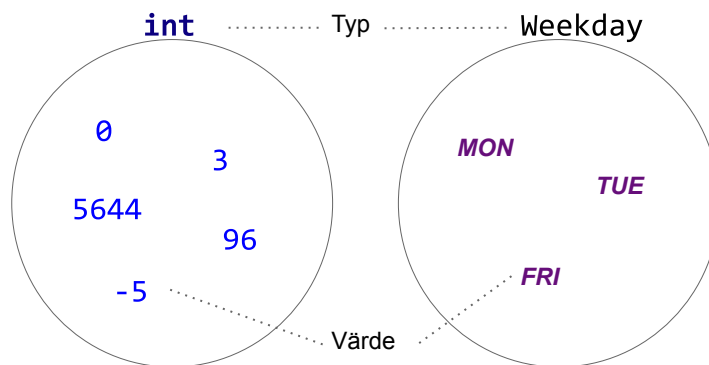
Vad är en Typ?



Finns inget enkelt svar.

- I denna kurs ser vi en typ som en mängd
- T.ex. mängden av reella tal, mängden av sanningsvärden o.s.v.

Typ och Värden



Ett värde är ett element ur en typ.

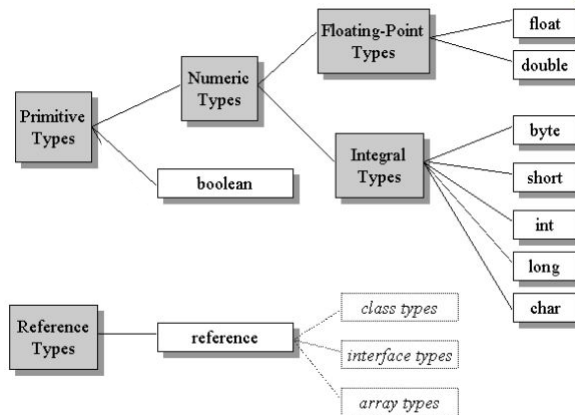
Typsystem

$$\begin{array}{c}
 \frac{x : \sigma \in \Gamma \quad \sigma \sqsubseteq \tau}{\Gamma \vdash_S x : \tau} \quad [\text{Var}] \\
 \\
 \frac{\Gamma \vdash_S e_0 : \tau \rightarrow \tau' \quad \Gamma \vdash_S e_1 : \tau}{\Gamma \vdash_S e_0 e_1 : \tau'} \quad [\text{App}] \\
 \\
 \frac{\Gamma, x : \tau \vdash_S e : \tau'}{\Gamma \vdash_S \lambda x. e : \tau \rightarrow \tau'} \quad [\text{Abs}] \\
 \\
 \frac{\Gamma \vdash_S e_0 : \tau \quad \Gamma, x : \tilde{\Gamma}(\tau) \vdash_S e_1 : \tau'}{\Gamma \vdash_S \text{let } x = e_0 \text{ in } e_1 : \tau'} \quad [\text{Let}]
 \end{array}$$

Exempel på "regler" för ett typsystem (inte Java's typsystem).

Ett typsystem är ett logisk regelverk för typer: Hur man kan härleda typen för uttryck och vilka operationer som är tillåtna för olika typer m.m.

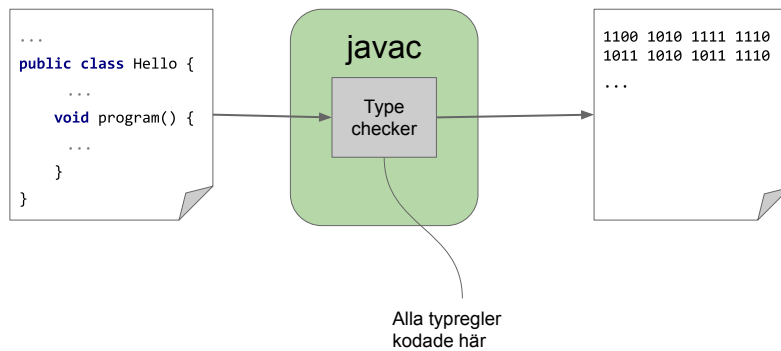
Typer i Java



Java har två principiellt olika typer: Primitiva och referenstyper

- De primitiva är fixa och färdiga, kan inte ändras eller lägga till/ta bort.
 - Har värdesemantik (kopior skapas)
- Referenstyperna kan byggas ut med egna typer t.ex. Dog, Player (egendefinierade typer).
 - Har referenssemantik..

Type Checker



6

Java är ett statiskt typat språk

- Java har ett typesystem som skall eliminera **typfel** redan vid kompileringen (d.v.s. **statiskt**, innan vi kört programmet).
 - I kompilatorn (javac) finns en "type checker" som sköter typkontrollen i samband med att koden kompileras.
 - Igen: Det skall inte kunna bli några typfel då vi kör programmet!
- För att typesystemet skall fungera måste alla värden i programmet ha en känd typ vid kompileringen.
 - Literaler ges automatiskt en typ, 123 får t.ex. typen int och true får typen boolean.
 - Uttryck kommer automatisk att ges en "beräknad" typ utifrån de ingående delarnas typer (literaler, variabler, operatorer, ...)
 - För variabler, arrayer och metoder måste vi explicit ange typen vid deklarationen, vi skriver helt enkelt int, boolean, Dog eller Player o.s.v. i koden.
 - Som sagt: Kompilatorn kommer i håg alla deklarationer (d.v.s. vet typen på allt) och ger ett fel om någon otillåten operation utförs på typen.

Varför Typer?



7

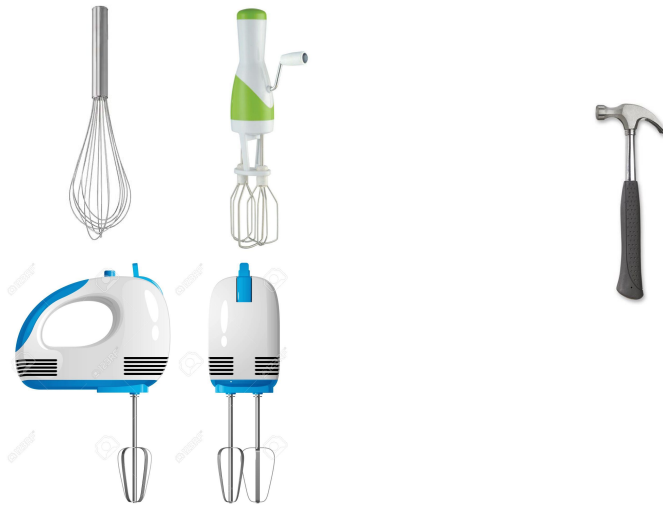
Typer används för att förhindra oss att använda data på ett felaktigt sätt, att blanda ihop saker och ting, att göra felaktiga (meningslösa) operationer.

- Typer är till för att hjälpa oss!
- ... men nybörjare tycker ofta de är i vägen ...

Analogi: Det skall vara salt i saltkaret och peppar i pepparkaret (och inget annat)!

En annan fördel med ett statiskt typsystem är också att typerna fungerar som en dokumentation, man ser tydligare vad man arbetar med och ev. hur det är tänkt att användas.

Super och Subtyper



8

Om vi var som helst i ett uttryck kan ersätta något av typ T med något av en annat typ S utan att något kompilersfel (typfel) uppstår säger vi att S är en **subtyp** till T (och T är en supertyp till S) skrivs $S <: T$.

Super/subtyp relationen ($<:$) är en binär relation mellan typer som säger att något ur en typ (subtypen), i alla sammanhang, kan ersätta något ur en annan typ (supertypen).

Super/subtyp relationen är transitiv

Super/subtyper gör att vi kan skriva mer generella program

Analogi: Antag att vi skall göra en sockerkaka. I receptet står, vispa ägg + socker... det står naturligtvis inte vilken sorts visp. Det går bra med vilken som! Alla vispar kan vispa. Däremot en hammare går inte att använda, den kan inte vispa

- Dvs: Det finns super/sub typ förhållande mellan vispar (de är utbytbara) men en visp och en hammare har inget super/sub-förhållande.

Super och Subtyper för Värden

```
// An expression that compiles (& and ~ are bitwise operators)
out.println((1 & 2) + 2.0 * 3.0 / ~4);

// Is it possible to replace any double value with an int value
// in the (any) expression and it will still compile?
out.println((1 & 2) + 2.0 * 3 / ~4);    // Replace 3.0. Ok!
out.println((1 & 2) + 2 * 3.0 / ~4);    // Replace 2.0 Ok!

// ... answer is : yes!

// Is it possible to replace any int value with a double and still
// compile? Answer is : NO! (bitwise operations not allowed for
double values)
out.println((1.0 & 2) + 2.0 * 3.0 / ~4); // Replace 1. Compile error
out.println((1 & 2) + 2.0 * 3.0 / ~4.0); // Replace 4. Compile error
```

9

Vi kan var som helst i ett uttryck där ett double-värde används ersätta detta med ett int-värde utan att typfel uppstår (se bild)


- D.v.s. int värde <: double värde
- Däremot kan vi inte ersätta int-värden med double-värden eftersom vissa operationer inte är tillåtna för double, vi får ett typfel
 - double är inte en subtyp till int, double är en **supertyp** till int.
- Det finns fler operationer för subtypen (int), vi kan göra minst lika mycket (mer) med subtypen (så är det alltid).

För primitiva typer gäller: char <: int <: float <: double, ger att char <: double

Implicita typomvandling

int double

12 + 4.0 -> 12.0 + 4.0 -> 16.0



Implicit typomvandling (int <: double)

```
out.println(1 + 2.0);    // 1 converted to 1.0 then +  
out.println(3.5 == 2);  // 2 converted to 2.0 then ==
```

```
out.println(true == 2); // No super/sub
```

10

Implicita typomvandlingar innebär en automatisk typomvandling av ett värde!

- Sker vid behova från från subtyp till supertyp
- Används för att göra det möjligt att blanda olika typer i t.ex. aritmetiska/booleska uttryck.
- Värden byter alltså typ

Finns inget super/sub kan ingen implicit typomvandling ske och vi får ett kompileringsfel.

ASIDE: För primitiva typer betyder det dessutom att bitarna i den binära representationen av värdet ändras.

Explicit Typomvandling

```
out.println(1 + (int) 2.0); // Ok. 2.0 to 2 then +  
out.println((int) 3.5 == 2); // Will lose precision
```

```
out.println(true == (boolean) 2); // No super/sub
```

11

Vi kan uttryckligen säga åt typsystemet att vi vill omvandla ett värde av en typ till en annan.

- Kallas **explicit typomvandling (typecasting, cast)**
- Kan bara ske om det finns en super/subtyp relation (med ett undantag senare).
- Parenteser kan behövas för att visa vad (vilket uttryck) som berörs
- Explicit typomvandling har högre prioritet än samtliga aritmetiska operatorer.

Super och Subtyper för Variabler

```
// An expression with variables that compiles
out.println((1 & i) + d * 3.0 / ~4); // i int and d double

// When variable used in expression it represents a value
out.println((1 & i) + i * 3.0 / ~4); // Replace d with i, OK!
out.println((1 & d) + d * 3 / ~4); // Replace i with d, NO!

// When variable on the left side of an assignment
// it does not represent a value. It represents a location!
// An assignment that compiles (d to the left).
d = (1 & i) + d * 3.0 / ~4;
// Try to replace with sub to the left
i = (1 & i) + d * 3.0 / ~4;
// An assignment that compiles (i to the left).
i = 1 / 2 + 3 * 4;
// Try to replace with super to the left
d = 1 / 2 + 3 * 4; // Ok!
```

12

Variabler kan anta två olika roller

- Till höger om = (tilldelningsoperatoren) eller i ett uttryck representerar de värden
- Till vänster i en tilldelning representerar de lagringsplatser (inte värden)

För variabler gäller

- På högersidan vid tilldelning eller i ett uttryck, ok att ersätta supertyp med subtyp enligt tidigare resonemang (värden).
- Vi kan inte ersätta variabel till vänster i en tilldelning med en variabel av någon subtyp!
 - Däremot kan den ersättas med en variabel av supertypen!

D.v.s. vi kan inte, i det generella fallet, ersätta en variabel av en viss typ med någon annan eftersom ersättaren inte kan vara super och subtyp samtidigt (beroende på position). Det finns ingen super/sub relation mellan variabler.

Super/sub vid Kompilering

`super = sub`

En variabel av supertyp
kan tilldelas ett värde
av subtypen

`sub = (sub) super`

Ett värde av
supertypen får
typomvandlas till
subtyp (och kan
därmed tilldelas
variabeln)

13

Det finns ingen super/sub mellan variabler

- Men ... utifrån föregående resonemang kan vi ställa upp de fundamentala krav kompilatorn ställer för att acceptera en tilldelning.

Subtyper för Klasstyper

```
class Player { ... }  
class Dog { ... }  
  
Player p = new Player();  
  
// No super/sub relation,  
// i.e. no implicit or explicit type casting  
Dog d = p; // No  
p = new Dog(); // No  
Dog d = (Dog) p; // No
```

14

Olika klasstyper har inte någon super/subtyp relation

- Det finns ingen implicit eller explicit typomvandling mellan olika klasstyper.
- Det går att skapa en super/subtyp relation, kommer senare.

Subtyper för Uppräkningsstyper

```
// Enumeration types
public enum WeekDay {
    MON, TUE, WED, THU, FRI
}
enum WorkingDay {
    MON, TUE, WED, THU, FRI;    // Same values ...
}

// No super/sub relation
WeekDay week = WeekDay.FRI;    // No
week = WorkingDay.FRI;        // No
week = (WeekDay) WorkingDay.FRI; // No
```

15

En uppräkningsstyp är en klasstyp d.v.s. ingen super/subtyp relation mellan uppräkningsstyper

- Implicita typomvandling till String (vid utskrifter).

Subtyper för Omslagstyper

```
Integer i;  
Double d;  
  
// int <: double but NOT Integer <: Double  
d = i;           // No  
i = d;         // No  
i = (Integer) d; // No  
d = (Double) i; // No
```

16

Omslagstyper är klasstyper d.v.s det finns ingen super/subtyp relation mellan omslagstyperna.

- Trots att `int <: double`!!

Typen Object

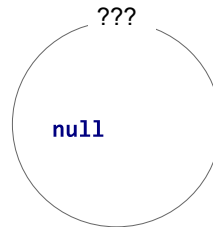
```
// Type Object is supertype to any reference type  
Object o;  
o = "Any";  
o = new Country();  
o = new int[]{1, 2, 3};  
o = 5;           // Boxing  
o = false;       // Boxing  
o = new Object();
```

Det finns en "översta" referenstyp, typen Object. Object är supertyp till alla referenstyper men har inte någon egen supertyp.

- D.v.s. det finns en klass Object färdig i Java (en klass introducerar en referenstyp)

null

```
// Nameless type for null is subtype to any reference type  
Object o = null;  
String s = null;  
Integer i = null;  
Number n = null;
```



null är enda värdet i en namnlös typ. Den namnlösa typen är subtyp till alla referenstyper.

- Namnlös typ d.v.s. kan inte deklarera variabler av typen.

Explicit Typomvandling för Referenser

```
// Dog <: Object and Player <: Object
// Compile ok, runtime ok
Object object = new Dog(); // super = sub
Dog dog = (Dog) object;    // In fact it was a Dog

// Compiles BUT ClassCastException runtime
// Can't assign dog object to player
Player player = (Player) object;
```

Vid explicit typomvandling mellan referenstyper kan det fungera vid körning ... eller inte.

- Som sagt: Typomvandlingen kopplar ur type checker:n, vi får ta ansvaret.
- Explicita typomvandlingar med referenser alltid farligt, undvik om möjligt.

ClassCastException

`super = sub`

Under körning måste
variabeltypen vara super
till objektets typ!

20

Under körning accepteras bara `super = sub`.

- Har vi gjort en felaktig explicit typomvandling för vi
ClassCastException vid exekvering.

Typ definierar Operationer

```
class Dog {  
    String name;  
    int age;  
}  
  
Dog dog = new Dog();  
dog.age++;           // Ok  
  
Object o = dog;  
o.age++;           // No age for objects!
```

21

Det är variabelns typ (statiska/deklarerade-typen) som avgör vad vi i fortsättningen kan göra med variabeln (tillåtna operationer).

- Variabeltypen definierar de tillåtna operationerna
- Om variabeln refererar ett subtyp objekt så är vissa operationer för subtypobjektet oåtkomliga

Subtyper för Primitiva Arrayer

```
// No subtype relations for primitive arrays
int[] ia = {1, 2, 3};
double[] da = {1.0, 2.0, 3.0};

//int <: double but *NOT* int[] <: double[]
ia = da;           // No
da = ia;           // No
da = (double[]) ia; // No!
```

22

Finns ingen super/subtyp relation mellan primitiva arrayer.

- Leder till hål i typsystemet, se nedan.

Subtyper för Referens-Arrayer

```
Integer[] iia = ...;  
Double[] dda = ...;  
iia = dda; // No  
dda = iia; // No  
iia = (Integer[]) dda; // No  
dda = (Double[]) iia; // No
```

Fall 1

23

Om elementtyperna inte har någon super/sub relation så har inte array-typerna någon super/sub.

Subtyper för Referens-Arrayer

Fall 2

```
Integer[] iia = {1, 2, 3};
```

```
// If Integer <: Object
```

```
// then Integer[] <: Object[]
```

```
Object[] os;
```

```
os = iia; // super = sub
```

```
iia = (Integer[]) os; // sub = (sub) super
```

24

P.g.a. vissa omständigheter*) bestämda de som utvecklade Java att för referenstyper gäller ...

- .. om $S <: T$ så är $S[] <: T[]$!!!
- T.ex. $\text{Integer} <: \text{Object}$ innebär att $\text{Integer}[] <: \text{Object}[]$

*) Java saknade från början generiska typer

The Array Loophole

```
Integer[] iia = {1, 2, 3};
```

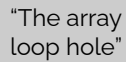
```
// If Integer <: Object then Integer[] <: Object[] !
```

```
// Will compile, but runtime exception!!
```

```
Object[] ns = iia; // Ok!
```

```
ns[0] = 4; // Ok! 4 is boxed to subtype Integer
```

```
ns[1] = 4.5; // ArrayStoreException!
```



"The array
loop hole"

25

The array loophole betyder att ett program som kompilerar (utan typomvandling) kan ge körningsfel. Något är fel! Typsystemet skulle ju garantera att det inte hände men

- Beror på att $S <: T$ så är $S[] <: T[]$ i Java
- Innebär att Java runtime alltid måste kontrollera vad vi stoppar in i en array (kostar tid)
- Försöker man stoppa in ett värde av felaktig typ får man en ArrayStoreException.

Suber/sub för Metoder

```
// Method to replace, which below will suffice?  
int doIt(int i) {  
    return i;  
}  
  
char doOther(double d) { // Return sub, param is super  
    return (char)(d + 1);  
}  
  
int doYetOther(char ch) { // Return same, param sub  
    return ch; // Implicit cast  
}  
  
double doYetYetOther(int i) { // Return is super, param same  
    return i; // Implicit cast  
}
```

26

Vi kan ersätta en metod med en annan metod om denna har subtypen till returtypen och supertypen för parametertypen.

Primitiva typer: Min och maxvärden

```
// Overflow
int i = Integer.MAX_VALUE;
out.println(i + 1);

// Underflow
int j = Integer.MIN_VALUE;
out.println(j - 1);

// True
out.println(Integer.MAX_VALUE + 1 == Integer.MIN_VALUE);

// False!!
out.println(Double.MAX_VALUE + 1 == Double.MIN_VALUE);
```

27

En mer implementationsmässig (teknisk) aspekt på primitiva typer är att de har olika storlekar i minnet och därmed min och maxvärden.

För Java gäller:

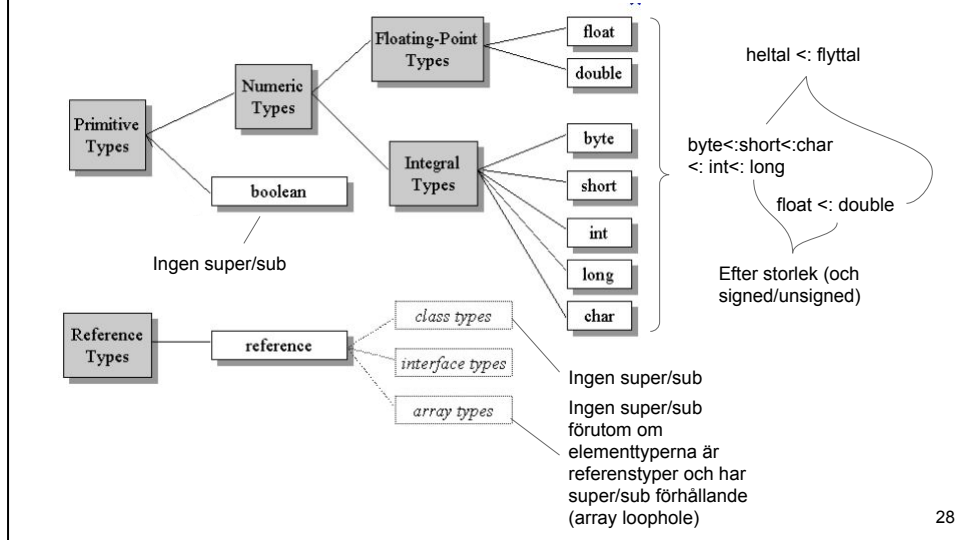
- int (32 bitar): -2147483648 - 2147483647
- char (16 bitar): 0 - 65,535
- double (64 bitar): 4.94065645841246544e-324d - 1.79769313486231570e+308d

Över eller underskrider vi max eller min värdena för en heltalstyp får vi overflow eller underflow.

- Innebär att värdet "slår över" från max till min och tvärtom.

OBS! Double.MIN_VALUE är inte negativt, det är det minsta värde som kan representeras, 4.9E-324.

Sammanfattning Super/sub (so far)



28

Super sub så långt vi kommit. Mer kommer.

- OBS! Super/sub för primitiva typer ganska ointressant för oss. Det är referenstyperna som är intressanta.