Subtyping och variance

Objekt-orienterad programmering och design

Alex Gerdes, 2018

Typer

- Java har två sorters typer – *primitiva typer* och *referens-typer*.
  - Primitiva typer är typer för *värden*: int, float etc.

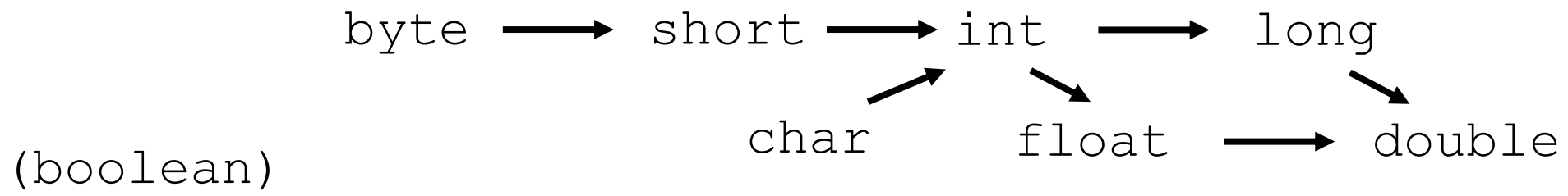
- Java har två sorters referens-typer – *array types* samt *class or interface types*.
  - Arrayer är ett specialfall. De bor på heapen, vi arbetar med dem via referenser, de är i praktisk mening objekt – men typerna ser lite speciella ut.

- Vi pratar här om *statiska* typer – de som används av typ-checkaren under kompilering.

Subtyper

- En typ A är en subtyp till en annan typ B, om alla element (värden eller objekt) av typ A kan användas som om de vore av typ B.
- En klasstyp C är en subtyp till
  - `Object`
  - Sin superclass (`extends`) och alla dess supertyper.
  - Alla de interfaces C implementerar (`implements`).

- För primitiver finns en hierarki:



Type conversion

- Typer kan på olika sätt *konverteras*:
  - (Värden med) primitiva typer kan konverteras till (värden av) andra primitiva typer.
  - (Värden med) primitiva typer kan konverteras till och från (objekt av) deras representativa klass-typer, e.g. `int` till `Integer`.
  - En referenstyp A kan konverteras till en supertyp B, e.g. `Triangle` till `Polygon`.
  - En referenstyp B kan *castas* till en subtyp A:
    - `Triangle t = (Triangle) polygons.get(0);`
    - Notera att detta enbart påverkar hur typen behandlas *statiskt*. Vi ber typcheckaren lita på oss – och skulle objektet vi arbetar med vid runtime ha fel typ får vi ett exception.

Conversions

- En konvertering till en ”bredare” typ kallas *widening conversion* och kommer alltid att fungera felfritt.
- En konvertering till en ”snävare” typ kallas *narrowing conversion*.
  - Med primitiver kommer viss loss of precision att ske, e.g. `long` till `int`.
  - Med referenstyper får vi fel vid runtime om objektet inte redan hade rätt *dynamisk* typ.

- En konvertering från e.g. int till Integer kallas *autoboxing*. En konvertering från e.g. Integer till int kallas *unboxing*.

Array vs ArrayList

- För arrays gäller:
  - Alla arraytyper är subtyper till Object.
  - En arraytyp A[] är en subtyp till typ B[] om A är en subtyp till B – arrayer är *kovarianta* (covariant).
  - Typen för elementen kommer att minnas vid runtime, och vi får fel om vi försöker göra dumt.

- För generiska typer gäller:
  - En generisk typ T<A> är *inte* en subtyp till typ T<B>, även om A är en subtyp till B – generiska typer är *invarianta* (invariant).
  - Typen för elementen kommer *inte* att minnas vid runtime, därför är den statiska typcheckaren mer strikt.

Generics och explicit varians

- För generiska typer kan vi explicit ange hur vi vill att subtyping ska hanteras:

Invariant

```
List<Polygon> = new ArrayList<Polygon>();
```

Covariant

```
List<? extends Polygon> = new ArrayList<Triangle>();
```

Contravariant

```
List<? super Triangle> = new ArrayList<Polygon>();
```

- Mer om varians (variance) och dess teori på nästa föreläsningen.

Övning

- Utgå från DrawPolygons sen tidigare (kan laddas ner färdig från hemsidan).
- Skapa en ny class TestSubtyping.
- Skapa några variabler av följande typer (med samma längd): `Polygon[]`, `Triangle[]`, `List<Polygon>`, `List<Triangle>`. Testa följande, och se vad kompilatorn säger. Försök förutspå resultaten först:
  - Vilken sorts objekt får ni lägga i respektive array eller lista? `Triangle`? `Polygon`? `Object`?
  - Mellan vilka av era variabler får ni lov att tilldela (dvs hela arrayer eller listor)?
  - Mellan vilka kan ni tilldela alla element från den ena till den andra (genom en for-loop)?
  - Vilken typ av objekt kan ni plocka ut ur respektive array (`a[index]`) eller lista (`l.get(index)`)?
  - Hjälper explicit casting vid något av ovanstående?
- Skapa två nya variabler av följande typer: `List<? extends Polygon>`, `List<? super Polygon>`. Gör samma tester som ovan (med listorna från ovan). Förutspå resultaten innan ni testar.
- Skapa en metod `paintAll(Graphics,List<Polygon>):void` som går igenom listan den får som argument och anropar `paint`-metoden på varje.
  - Vilka av ovanstående listor kan ni ge som argument till denna metod?
  - Kan ni lägga till nya element i listan inuti metoden?
- Ändra parametertypen för `paintAll` till först covariant och sen contravariant. Hur påverkar respektive alternativ? Förutspå resultaten!
- Skapa en metod som tar två listor som argument, och flyttar alla element från den första till den andra. Vilka typer bör ni ge parametrarna för att göra metoden så användbar som möjligt?
  - Kan ni göra samma sak för arrays? Hur skiljer det sig?