Subtyping, interfacce, ereditarietà (Java)

Interfaccia

`interface I {...}` `class C implements I {...}`

Specifica astratta dei metodi pubblici che una classe deve implementare per soddisfare l'interfaccia. Contengono solo il prototipo dei metodi, e sono utilizzate per modellare ADT (abstract data type). Una classe può implementare più interfacce.

Ereditarietà

`class B extends A {...}`

`B` eredita da `A` tutti i membri (variabili e metodi) non `private` e che non sovrascrive.

1. ereditarietà singola/semplice;
2. le classi che non estendono nessuna altra classe sono implicitamente sottoclassi di `Object`, che contiene metodi come `toString`, `equals` e `clone`;
3. si può fare *overriding* di metodi con stesso prototipo.

Java supporta le *classi astratte*, classi in cui almeno un metodo non contiene l'implementazione. Possono solo essere estese (non istanziate direttamente), e vengono utilizzate come interfacce che forniscono anche variabili e implementazioni di alcuni metodi.

Subtyping

Java implementa polimorfismo per sottotipo:

- se `A implements B` o `A extends B`, allora `A` $<:$ `B` – ogni membro pubblico di `B` è anche membro pubblico di `A`
- *nominal subtyping*: la relazione deve essere esplicitata sintatticamente;
- il compilatore verifica comunque la coerenza strutturale tra `A` e `B`;
- `Object` è il tipo *top* della gerarchia dei sottotipi.

Conversioni: dato $A <: B$,

- la regola di subsumption viene applicata implicitamente quando necessario per convertire oggetti di tipo A in oggetti di tipo B (upcast);
- se il tipo statico di un oggetto è B ma il tipo effettivo è A, si può effettuare un downcast esplicito da B ad A (che genera un errore a runtime se il tipo effettivo non è corretto).