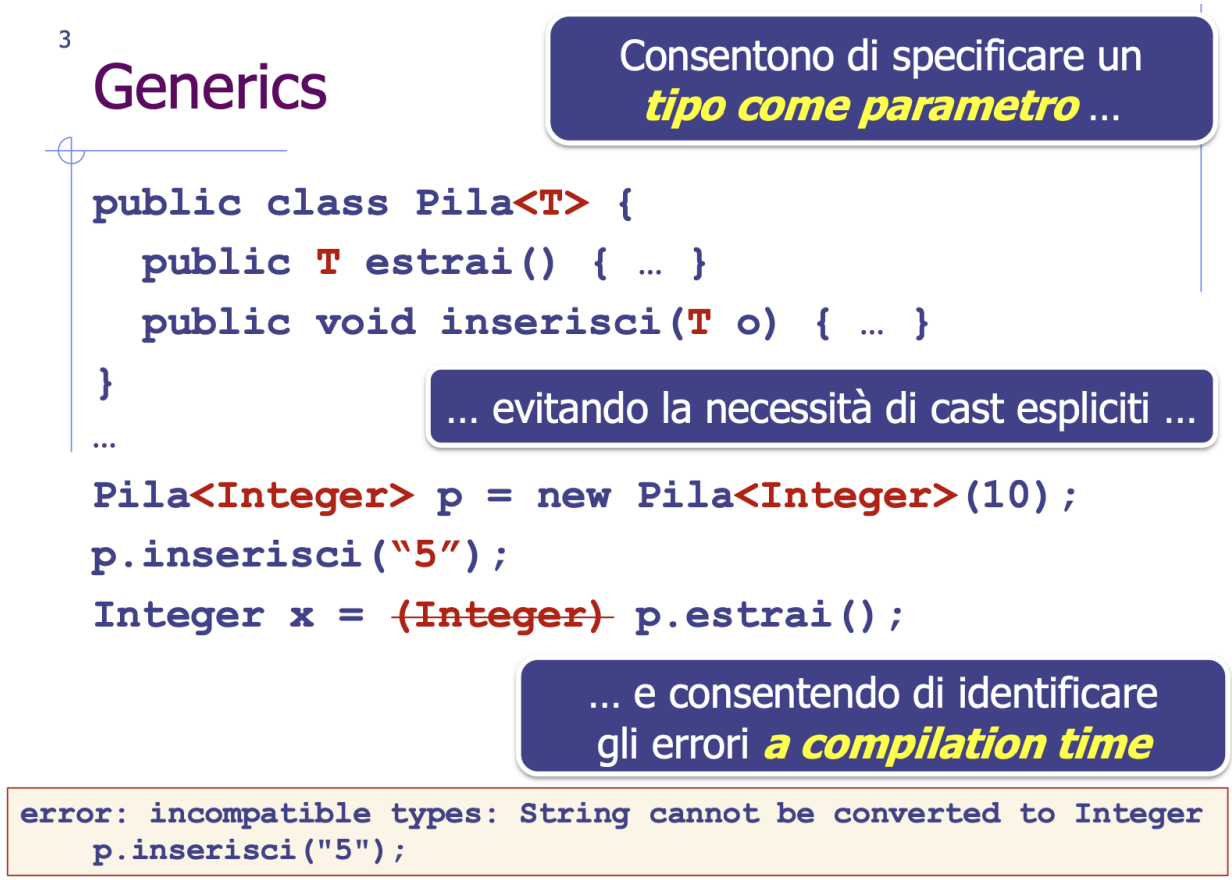
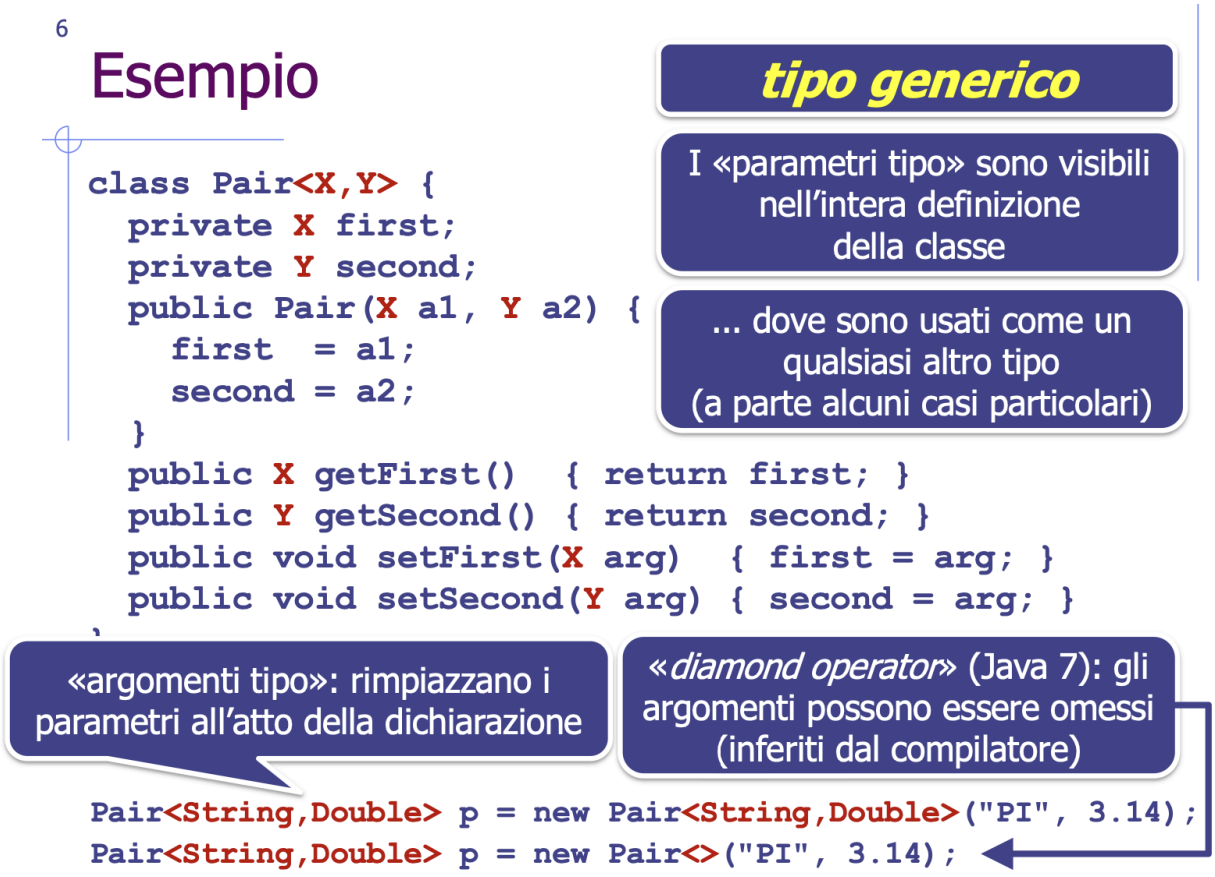
Generics

Nel caso in cui si volessero creare strutture dati polimorfe, la richiesta di **cast e conversioni esplicite** sarebbe molto elevata e potrebbe portare a **rischi di errori rilevati solo a runtime**. A questo proposito entrano in nostro aiuto i **generics**. I generics consentono di specificare un tipo come parametro evitando quindi la necessità di utilizzare cast espliciti. Così facendo, oltre ad abbassare il rischio di errori, viene fatto sì che eventuali errori vengano identificati a **compilation time**.



A livello sintattico vengono utilizzati nel seguente modo: " class name <T1, …, T_n> " dove T1,...T_n rappresentano i **tipi con i quali la classe è parametrizzata**. Nel momento dell'utilizzo dei metodi definiti con i parametri tipo generici, i parametri tipo vengono rimpiazzati con parametri a scelta.



Generics e Array

Un aspetto importante e non da sottovalutare con i Generics è il fatto che **non è possibile creare un array generico**. Implica quindi che tutte le espressioni di questo tipo sono **illegali**:

- new Group<T> [...]
- new Group<Persona> [...]
- new T [...]

Il motivo della loro illegalità sta nel fatto che così facendo il linguaggio non sarebbe più *typesafe* e quindi minerebbe il vantaggio primario dei Generics. Segue che in generale in questi casi è meglio usare le **List,** più flessibili e generiche.

Ulteriori limitazioni

Si evidenzia che utilizzando i generics, i parametri tipo **non possono essere** tipi primitivi, si è costretti ad operare casomai con *Wrapper e Autoboxing*: Pila<int> → VIETATO!
Infine è necessario far notare che non si possono utilizzare i Generics con il comando **instanceof**.