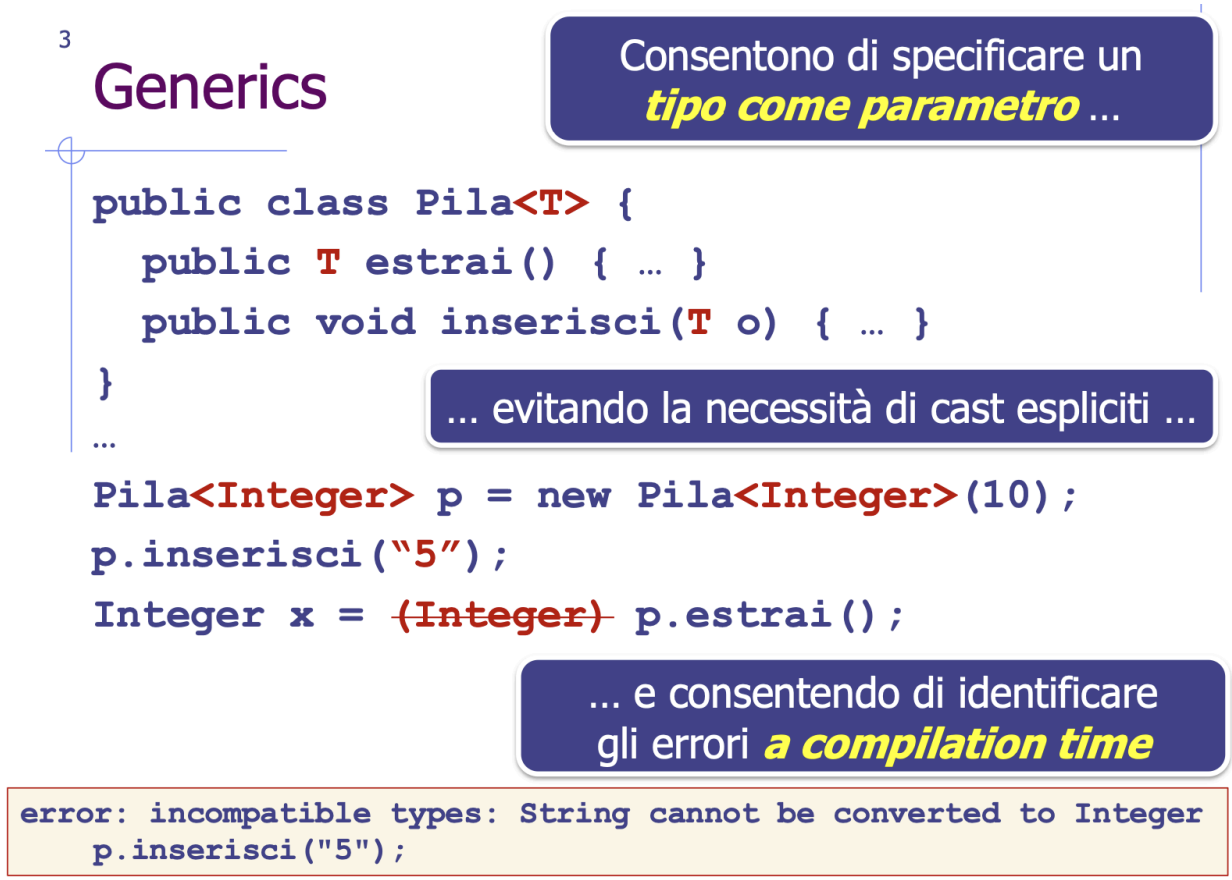
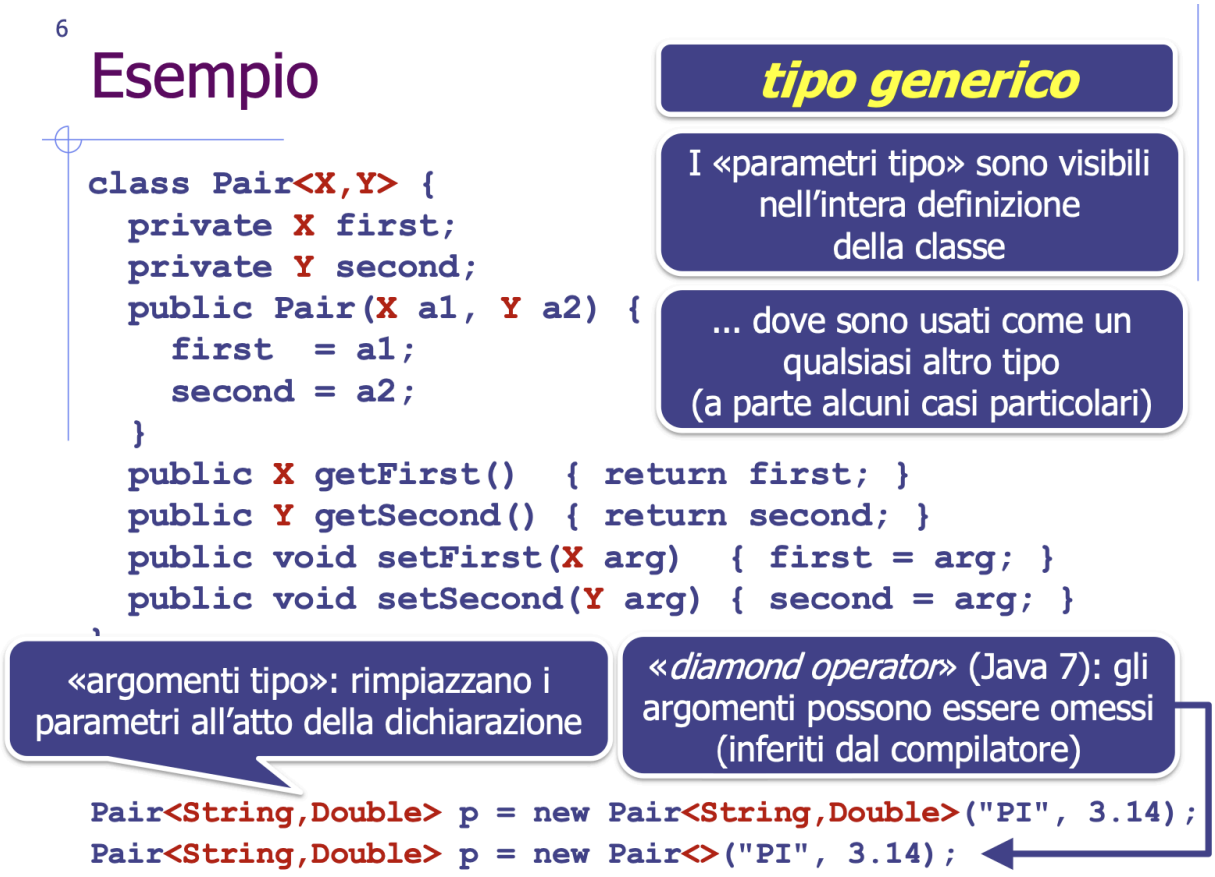
Generics

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* (forti problemi di casting in quanto Java non sa esattamente quello che è contenuto negli array) 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**.

Esempi pratici di utilizzo dei Generics

Bounded Generics

Ipotizzando di voler creare una classe *Generics* aggiungendo però una restrizione: qualsiasi sia il tipo generico che verrà passato deve essere un sottotipo di una determinata classe. Chiamiamo questa dinamica **bounded generics**.

Nell'esempio riportato viene creata una classe addetta alla stampa con la restrizione che vengano stampati solo animali (qualsiasi tipo di animale) quindi **qualsiasi classe** a patto che **estenda da un animale**.

```
public class Printer <T extends Animal>{

  T thingToPrint;

  public Printer(T thingToPrint) {
    this.thingToPrint = thingToPrint;
  }

  public void print() {
    System.out.println(thingToPrint);
  }
}
```

Si noti che essendo **T** un tipo che estende **Animal**, all'interno della classe **Printer** potremmo chiamare tramite **T**, tutti i metodi della classe **Animal** in quanto vengono ereditati.

Metodi Generics

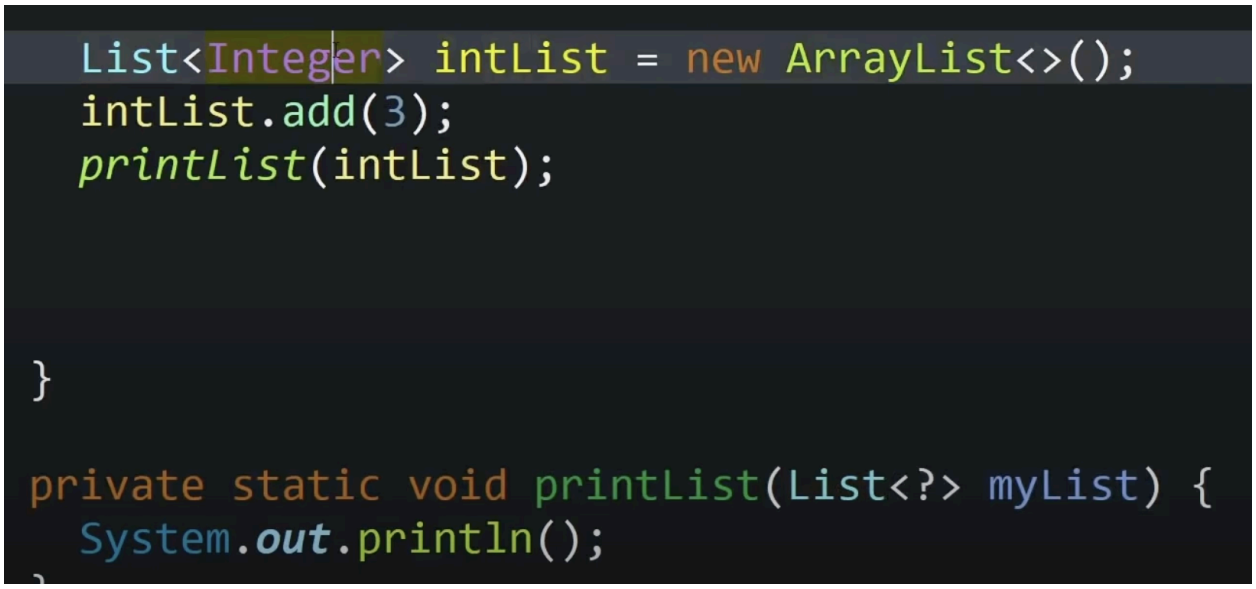
Si ipotizzi di voler creare un metodo che accetti come tipo i *generics*, se lo si scrivesse come normalmente si scriverebbe un metodo che opera su un parametro passato, il compilatore ci farebbe notare che **non esisterebbero classi con quel nome**.

```
private static <T> void shout (T thingToShout) {
  System.out.println(thingToShout + "!!!!!");
}
```

Si deve invece, prima del tipo ritornato dal metodo, **specificare che si opera con un Generics**.

Wildcards

Il vantaggio delle wildcard arriva quando dobbiamo operare con strutture dati di tipo **List** che possono accettare solo elementi generici. Supponiamo di voler passare come parametro ad un metodo che stampa una lista, una lista di elementi qualsiasi. Intuitivamente verrebbe da passare una lista di *Objects* ma se poi nel main si tenterebbe di chiamare questo metodo passando come parametro una lista di un qualsiasi tipo (che di fatto è una sottoclasse di Object) si otterrebbe un errore in fase di compilazione. Questo perché verrebbe accettata solo ed esclusivamente una lista di tipo Object come parametro. Vale quindi la seguente regola: **se B è sottoclasse di A**, **non vale assumere che List<B> sia sottoclasse di List<A>**. questi due elementi **non hanno alcuna correlazione per il compilatore**. A questo punto sfruttiamo le Wildcards:



Le Wildcards si indicano con il **<?>** e ci specificano che il tipo di liste passato può essere qualsiasi cosa, risolvendo il problema delle sottoclassi.

In aggiunta alle Wildcards possono essere anche create le *bounded wildcards* che permettono di restringere ulteriormente il campo di tipi accettati **nel caso di utilizzo di Liste** o **strutture dati derivanti da List**.