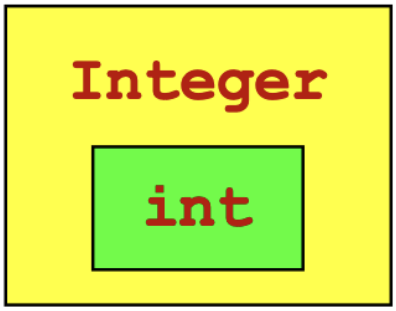
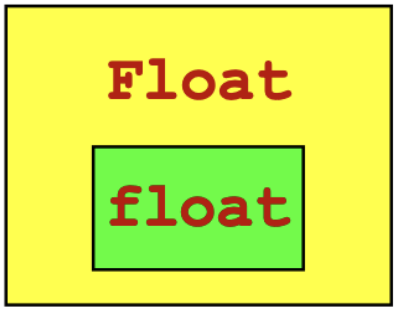
Classi Wrapper

Le classi dette “Wrapper” sono classi native del linguaggio Java e servono per **generare oggetti** che contengono al loro interno un **tipo di dato primitivo**. La denominazione è semplicemente il nome del tipo primitivo ma con la notazione di sintassi delle classi Java, con la prima lettera maiuscola.





Questi strumenti risultano essere estremamente utili quando si deve operare con metodi che prendono in input Oggetti generici ma concettualmente si sta lavorando con numeri.

Conversioni automatiche

Analizziamo un’espressione del tipo: “ **s.inserisci(k)** ” dove “ **s** “ è una collezione di Objects, “ **k** “ un intero, mentre il metodo “ **inserisci (Object o)** “ prende come input un oggetto di tipo Object e lo aggiunge alla collezione. Per quello che abbiamo accennato delle Classi Wrapper, all’interno dell’espressione andrebbe posizionato un *cast* con il compito di trasformare l’intero in una classe Wrapper che essendo una classe di Java è sottoclasse di Object e per il polimorfismo può essere passata al metodo “ **inserisci** ”. Questo processo, invece, è automatizzato e viene operato da Java stesso nella sua versione superiore o uguale a 5. Tale conversione viene chiamata **conversione automatica** e semplifica molto la vita del programmatore. Se questa conversione è dal tipo base alla classe Wrapper allora è chiamata *auto-boxing* altrimenti se il processo è al contrario viene chiamata *auto-unboxing*.

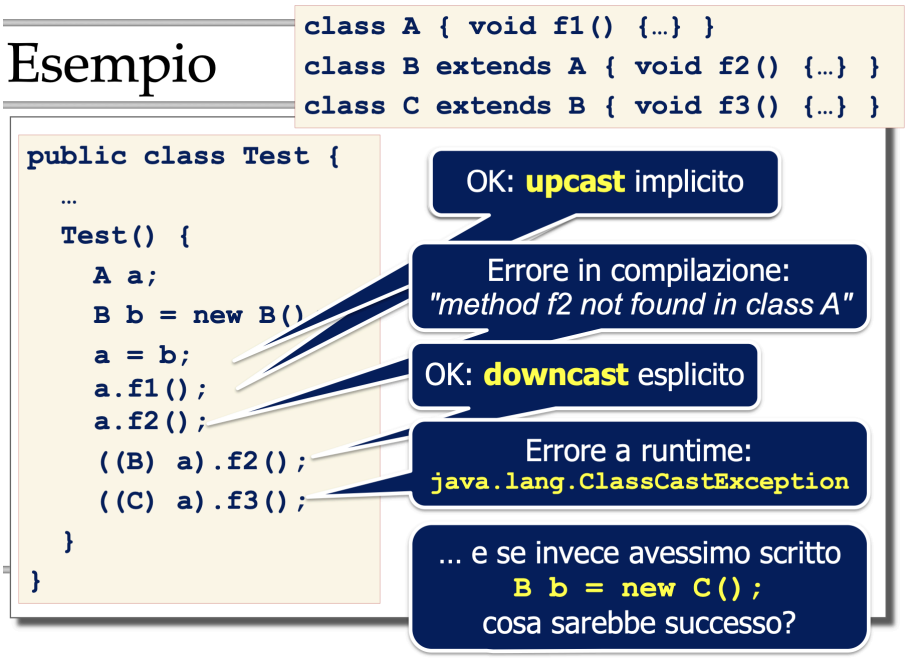
Si presti attenzione al fatto che le conversioni automatiche **non operano proprio su tutte le conversioni tra Classi Wrapper**, alcune necessitano di un casting esplicito, ne è un esempio: “ **int w = s.estrai( )** “, è un’espressione sbagliata perché necessiterebbe di un cast “ **(Integer)** “ o addirittura “ **(int)** “ prima di “**s.estrai( ) “**.

Casting non automatico

Come evidenziato in precedenza è possibile fare un *casting* **forzato**, ovvero una conversione tra tipi di riferimento differenti (ipotizziamo il passaggio da T ad un sottotipo T1). Ciò può avvenire solo a patto che **il tipo dinamico dell’oggetto convertito sia un sottotipo di T1**.

```
Object o = new AutomobileElettrica();
Automobile a = o; // errato, Object non è un
                  // sottotipo di Automobile
Automobile a = (Automobile) o; // corretto (casting)
```

In questo caso “Object o” rappresenta l’oggetto **T** mentre “Automobile a” rappresenta il suo sottotipo **T1**. Esistono due tipi di operazioni di *casting*: l’**upcast** e il **downcast**, questa versione esplicita prende il nome di downcast, mentre la conversione implicita effettuata dal polimorfismo prende il nome di upcast.



Si noti che gli errori di *casting*, essendo legati al tipo dinamico di un'istanza, vengono generalmente identificati nella fase *runtime*, sono quindi ERRORI RUNTIME: *java.lang.ClassCastException*.
Per determinare il **tipo dinamico** possiamo avvalerci di un utile metodo della classe Object che è **instanceof**, utile per evitare errori runtime dovuti a downcast espliciti effettuati dall'utente.