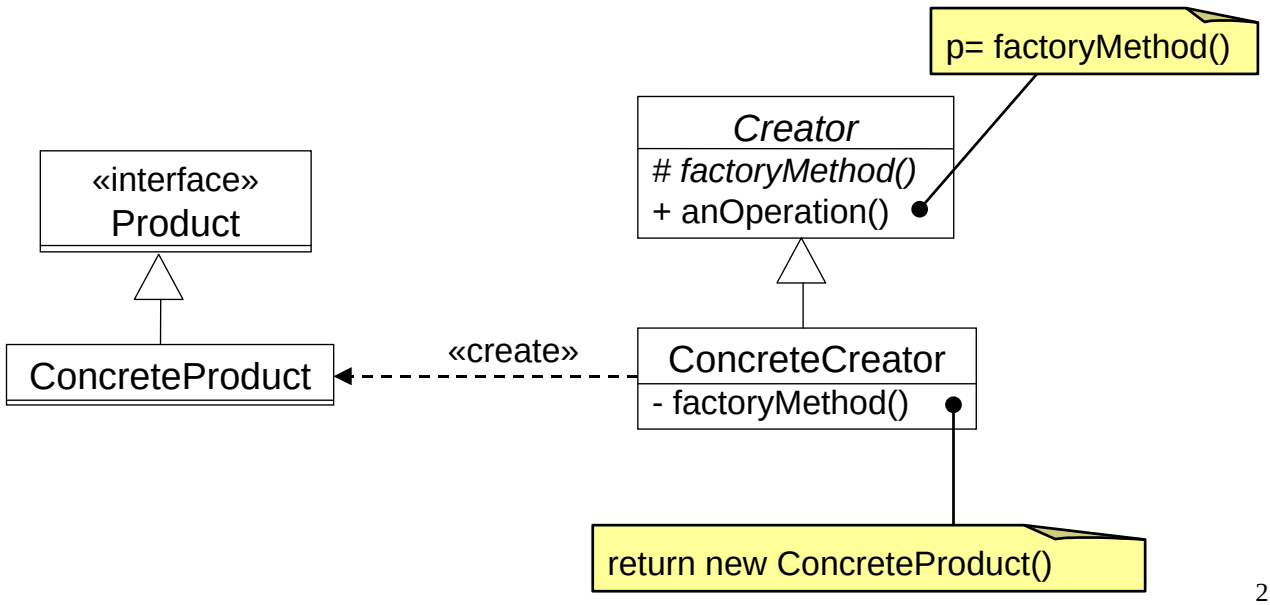
# design patterns (GoF)

Factory Method, Iterator



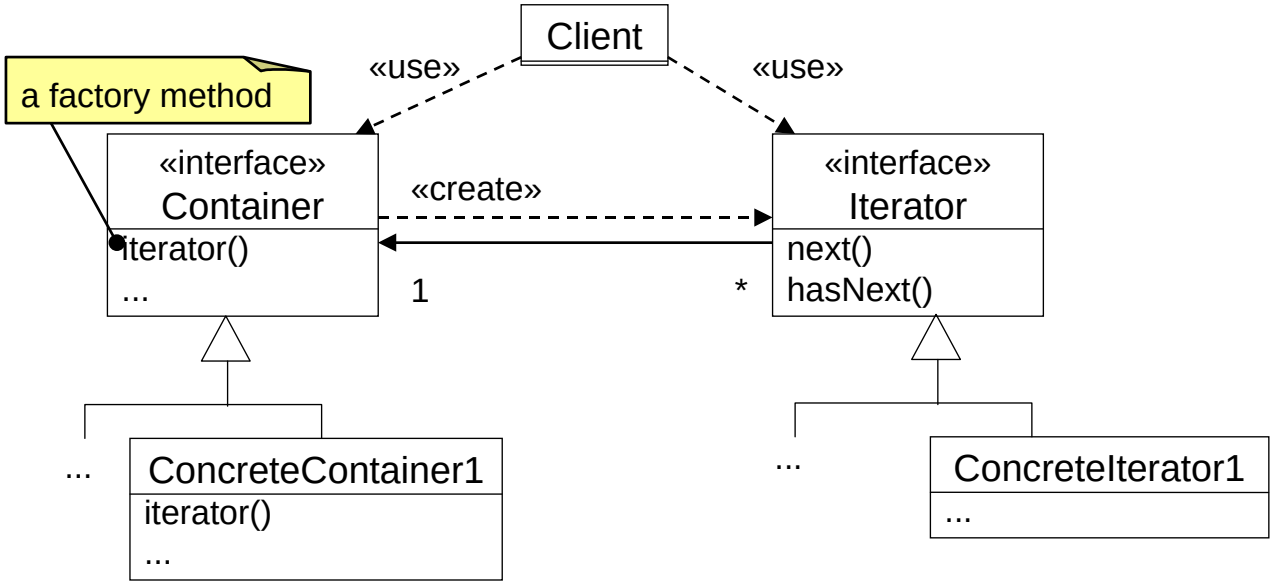
# Factory Method

- intento
  - definire una interfaccia per la creazione di un oggetto ma lascia alle sottoclassi la decisione sul tipo dell'oggetto.
- struttura



# Iterator

- intento
  - fornire un modo per accedere agli elementi di un aggregato senza esporre l'implementazione sottostante
- struttura






# Iterator: consistenza

- lo stato di Iterator deve essere consistente con quello di Container
  - la natura della consistenza dipende dalla implementazione di Container e Iterator
- un client può introdurre inconsistenze...
  - ...distruggendo un Container su cui si hanno degli Iterator attivi (se ammesso da linguaggio)
  - modificando il Container mediante i suoi metodi modifier
    - cancellando elementi
    - o forse anche aggiungendo elementi
    - modificandone l'ordine (se questo ha un senso per il container)

# esercizio: safe iterator

- modificare il pattern iterator in modo che gli iteratori siano “sicuri”

  due approcci:

  - invalidazione degli iteratori
  - locking dei contenitori
- un iteratore sicuro viene **invalidato** quando Client invoca una operazione **non ammessa** su Container
- un Container viene lockato quanto si crea un iteratore e slockato quando l’iteratore viene distrutto
- sono possibili approcci misti, c’è spazio almeno per tre progetti diversi!



# esercizio: obiettivi del progetto

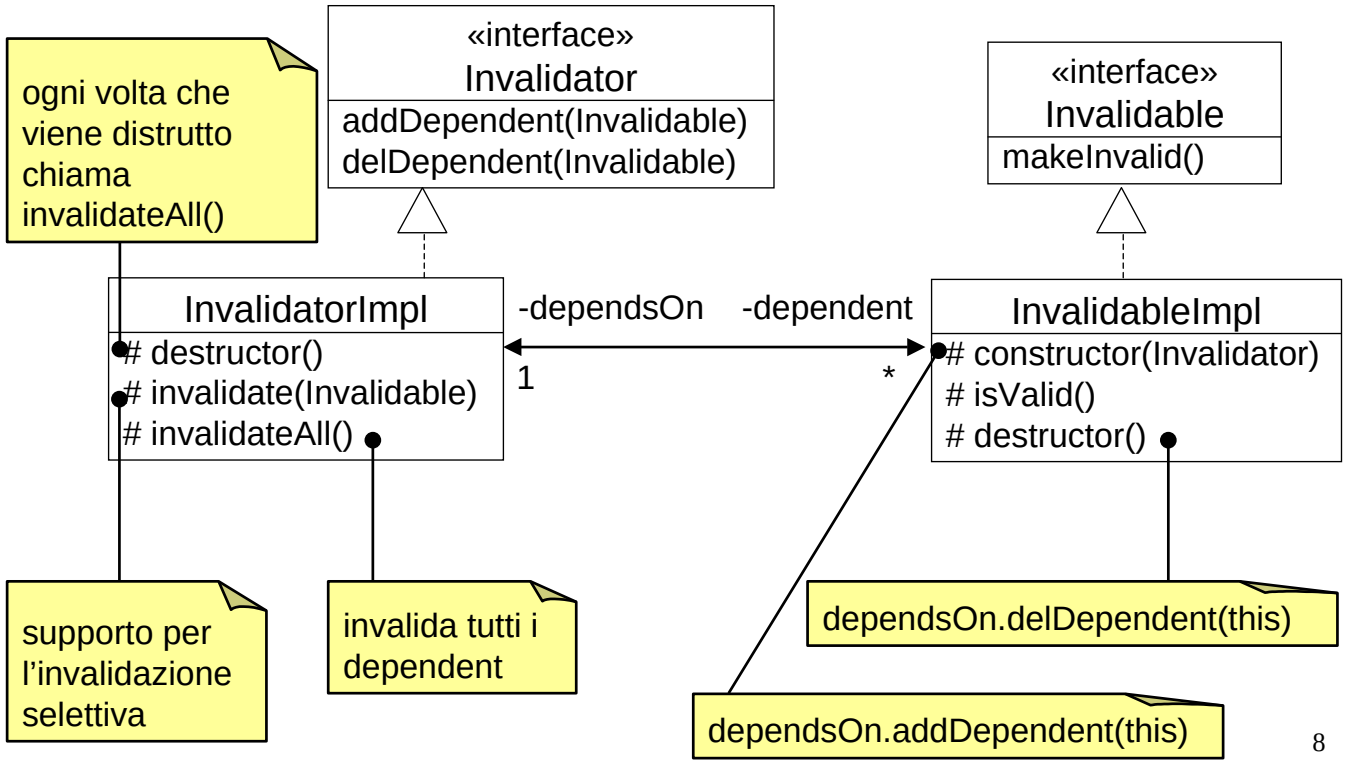
- uno degli obiettivi dovrebbe essere di mascherare all’utente il meccanismo dell’invalidazione e/o del locking
- rendere disponibile un metodo per riutilizzare i meccanismi di dell’invalidazione e/o del locking per altre implementazioni di coppie Container/Iterator
- attenzione
  - tale meccanismo potrebbe essere utile per vari tipi di Container

# esercizio: problematiche da considerare

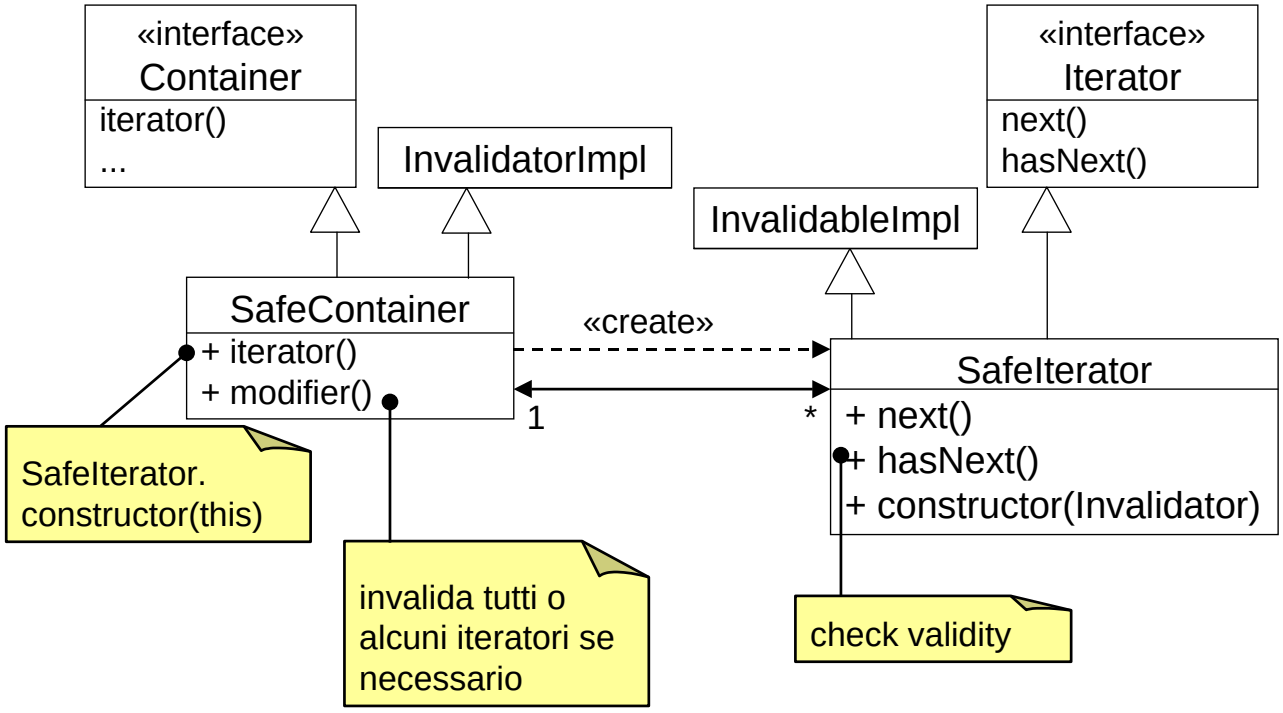
- alcune operazioni di Container invalidano tutti gli Iterator
  - Container deve avere conoscenza di tutti gli Iterator validi
- costruzione di un Iterator
  - deve aggiornare il Container
  - viene effettuata dal Container → non ci sono problemi
- la distruzione di un Iterator
  - deve aggiornare il Container
  - viene effettuata da un client tale logica deve essere inserita in un distruttore
  - un Iterator deve conoscere il suo Container



# supporto all'invalidazione



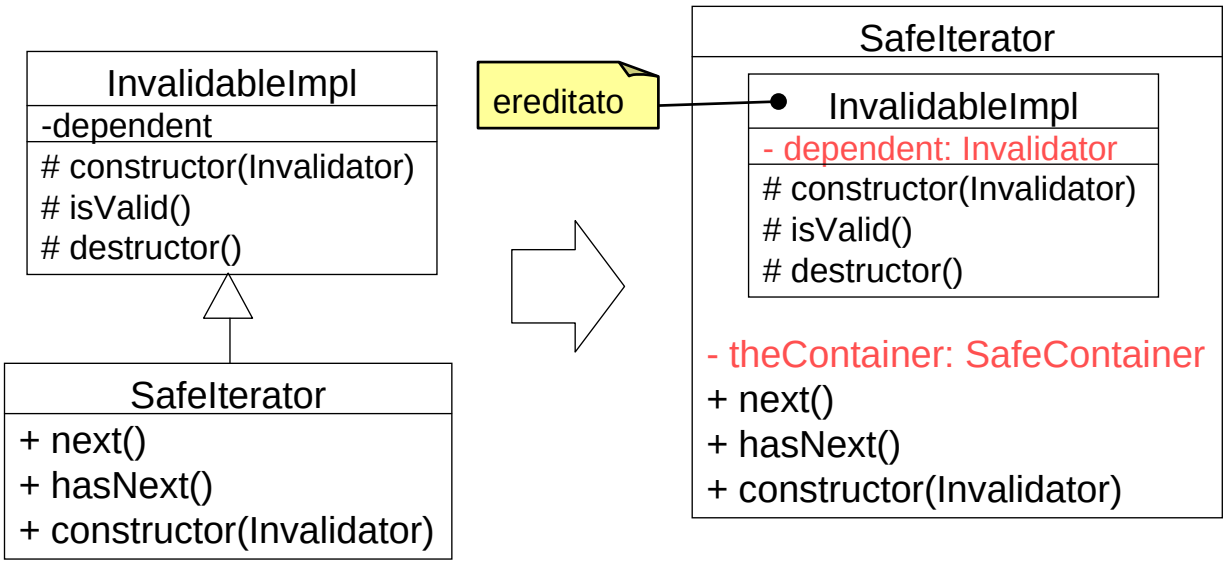
# iteratori sicuri






# osservazione/esercizio

- questo progetto introduce una ridondanza, quale?




- modificare il progetto di Invalidable in modo da eliminarla poter creare SafeIterator senza ridondanza

# categorie di pattern

- i pattern vengono suddivisi in categorie:
- creational
  - relativi alla creazione di oggetti
  - es. factory method
- structural
  - relativi alla struttura della applicazione
  - es. adapter, decorator
- behavioral
  - relativi al comportamento
  - es. iterator, observer



# altri pattern importanti

- creational
  - singleton: classi che ammettono una sola istanziazione
    - es. java.lang.System
- structural
  - facade: interfaccia unificata a complessi sistemi di classi
    - es. interfaccia semplice ad un compilatore:
      vedi in java/lib/tools.jar la classe com.sun.tools.javac.Main
  - composite: oggetti composti senza “limiti di profondità”
    - es. un filesystem con subdirectory

# altri pattern importanti

- behavioral
  - command: oggetti che rappresentano operazioni
    - es. undo
  - chain of responsability: decisione dinamica su chi deve servire una certa richiesta in un composite
    - es. help contestuale
  - state: oggetti che cambiano comportamento di tutti i metodi secondo il loro “stato”
    - es. TCP connection (metodi, open, close, read, write, ecc.)
  - strategy: algoritmi come oggetti
    - es. vari algoritmi di compressione usabili allo stesso modo
  - template method: scheletri di algoritmi customizabili
    - es. visite di alberi