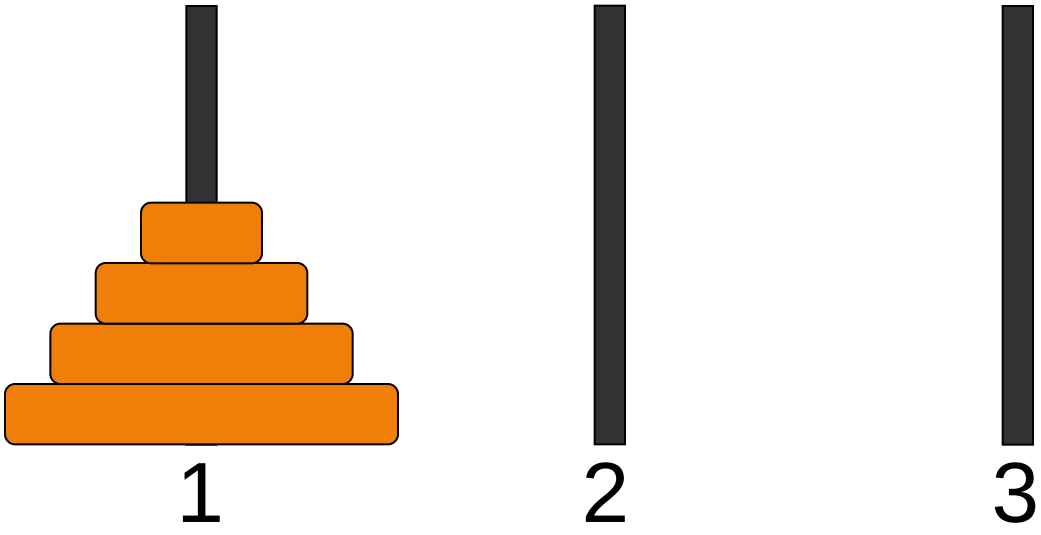
Esercizio: torre di Hanoi

- Il gioco della “Torre di Hanoi” è un rompicapo dalle proprietà matematiche interessanti.
- Il gioco prevede tre postazioni dove impilare dischi di diametro differente.
  - Si parte con tutti i dischi nella postazione 1, bisogna finire con tutti i dischi nella postazione 3.
  - Ad ogni mossa si può spostare il disco in cima alla pila da una postazione a un’altra (in LIFO).
  - Non si può mai mettere un disco più grande sopra uno più piccolo.

Esercizio: torre di Hanoi



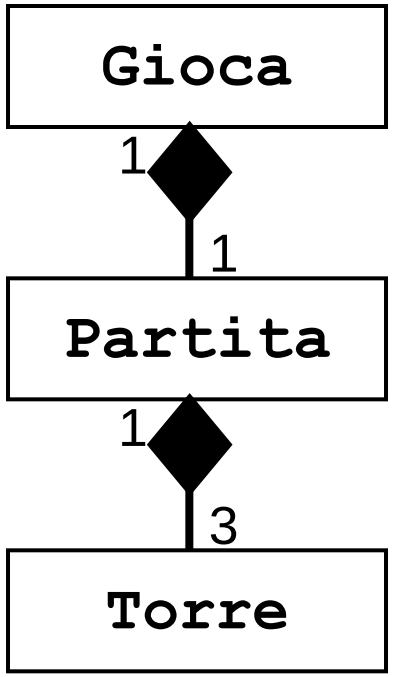
- Il problema è quello di compiere lo spostamento nel numero minore di passi.
  - La sua soluzione si ottiene con un algoritmo che, per quanto facile da generalizzare a torri molto grandi, richiede un tempo esponenziale.

Esercizio: torre di Hanoi

- Abbiamo bisogno di tre oggetti “Torre” che hanno politica di accesso a pila.
  - Possiamo usare array di dimensione fissata (per una versione più semplice): nulla vieta di usare però anche una struttura dati più complessa.
- La dimensione massima della pila è uguale al numero di dischi in gioco (quando i dischi sono tutti su una torre).
  - Parametro richiesto all’utente all’inizio del gioco, in un range tra 3 e 10 (suggerito come massimo)

Esercizio: torre di Hanoi

- Struttura delle classi:
  - **`Gioca`** è la classe di prova, chiede quanti dischi vogliamo e poi creare una partita in loop.
  - **`Partita`** inizializza le tre torri, poi chiede input all'utente per muovere i dischi.
  - **`Torre`** è la struttura dati, e ogni partita ha esattamente 3 torri.



Esercizio: torre di Hanoi

- Suggerimenti:
  - Iniziare scrivendo l'implementazione di **`Torre`**: essendo una pila, deve avere un metodo push, un metodo pop, e inoltre un metodo di visualizzazione (una specie di **`toString`**).
  - Ad esempio, si potrebbe procedere riga per riga, dando in ingresso al metodo il valore della riga come parametro, e ottenendo in risposta: " **`I`** " se non c'è niente in quella riga; "**`XXXXX`**" se in quella riga c'è un disco (calcolando quante **`X`** ci vogliono)

Esercizio: torre di Hanoi

- Poi **`Partita`** ha un metodo principale che visualizza le torri e chiede input all'utente sulla mossa da eseguire (finché non è finita):
  - se viene digitato un intero 1-3 chiede un altro intero e sposta i dischi in modo corrispondente
  - se viene digitato un certo comando (ad es. 0) chiede conferma che si vuole finire la partita
  - tutte queste letture di input richiedono di catturare eventuali eccezioni non controllate che si verificano se l'input fornito non è idoneo (ad esempio si mette una stringa)

Esercizio: torre di Hanoi

- In realtà le eccezioni relative al testo mal formattato sono **eccezioni non controllate**.
- Vogliamo però che i metodi push e pop della pila rispettino certe regole:
  - non si può fare pop da una pila vuota
  - non si può fare push di un disco più grande sopra uno più piccolo
- Per rispettare queste regole, introduciamo opportune **eccezioni controllate**.

Esercizio: torre di Hanoi

- Possibile implementazione:

```
public class TowerException extends Exception
{ public TowerException()
  { super(); }
}
public class TPushException extends TowerException
{ public TPushException()
  { super(); }
}
public class TPopException extends TowerException
{ public TPopException()
  { super(); }
}
```