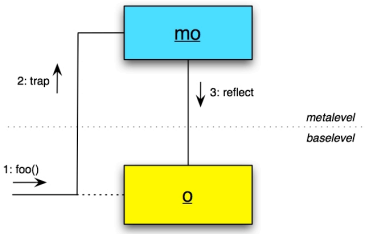
Riflessione Computazionale

- La riflessione è la capacità di un sistema (detto riflessivo) di contenere strutture che rappresentano *aspetti* di sé stesso (*metadata*) che gli permettono di supportare azioni su sé stesso
- Un sistema riflessivo è rappresentabile come un sistema a livelli in cui il livello sottostante, il *livello base*, non conosce che esistono livelli superiori, *livelli meta*
- Un livello superiore è in grado di ridefinire alcune operazioni del livello sottostante
  - Per cambiarne la natura
  - Per inserire nuove funzionalità
- Il livello superiore è capace di *intercettare* le operazioni del livello inferiore ed *ispezionare* il livello inferiore

Riflessione Computazionale

- La riflessione permette di costruire programmi che
  - Ispezionano strutture e dati di programmi
  - Prendono decisioni in base ai risultati dell'ispezione
  - Cambiano il comportamento, la struttura o i dati del programma in base alle decisioni prese



Esempio 1: invoke()

- Vogliamo invocare un metodo show() su una istanza di cui non conosciamo la classe (perché sviluppata da terze parti)
- Senza riflessione, dobbiamo usare un costrutto per l'invocazione per ogni specifica classe. Conseguenze:
  - Codice di invocazione dipendente dalla classe
  - Codice da modificare ogni volta che è prodotta una classe
- Con la riflessione, l'invocazione è resa indipendente dalla classe a cui il metodo show() appartiene

```
public static void showObject(Object o, String msg) {
   ...
   // introspezione
   Class cls = o.getClass(); // cls rappresenta la classe
   Method m = cls.getMethod("show", new Class[] {String.class});
   // invocazione dinamica
   m.invoke(o, new Object[] {msg});
   ...
```


Invocazione Dinamica

- Tramite invoke(..) riusciamo a chiamare un metodo di un oggetto a runtime senza specificare a design time di quale metodo si tratti
- A design time, possiamo non specificare
  - Nome metodo
  - Nome classe

Esempio 2: invoke()

- Vogliamo invocare i metodi get*() su una istanza di cui non conosciamo la classe e scrivere (su disco) i valori di ritorno del metodo chiamato

```
public static void getValues(Object o) {
   Class cls = o.getClass();

   Method mt[] = cls.getDeclaredMethods();

   for (int i = 0; i < mt.length; i++) {
      if (mt[i].getName().startsWith("get") )
      try {
         Object v = mt[i].invoke(o, null);
         ...
```


Esempio forName()

- Vogliamo caricare una classe di cui conosciamo il nome solo a runtime (non a design time) e creare una sua istanza
  - Al fine di invocare su tale istanza un metodo

```
// carica una classe e ritorna un corrispondente oggetto
Class cls = Class.forName( nomeClasse );

// crea una istanza di una classe
Object = o cls.newInstance();
...
m.invoke(o, null); // invoca un metodo dell'istanza creata
```


Caricamento Dinamico

- Tramite forName(..) riusciamo a caricare una classe a runtime senza specificare a design time di quale classe si tratti

- Tramite newInstance(..) riusciamo a creare un oggetto della classe che conosciamo solo a runtime