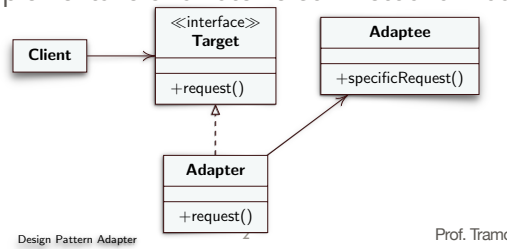
Design Pattern Adapter

- **Intento**: Converte l'interfaccia di una classe in un'altra interfaccia che i client si aspettano. Adapter permette ad alcune classi di interagire, eliminando il problema di interfacce incompatibili
- Problema
  - Alcune volte una classe di una libreria non può essere usata poiché incompatibile con l'interfaccia che si aspetta l'applicazione. Ovvero nome metodo, parametri, tipo parametri di chiamate all'interno dell'applicazione non sono corrispondenti a quelli offerti da una classe di libreria
  - Non è possibile cambiare l'interfaccia della libreria, poiché non si ha il sorgente (comunque non conviene cambiarla)
  - Non è possibile cambiare l'applicazione, e si può voler cambiare quale metodo invocare, senza renderlo noto al chiamante

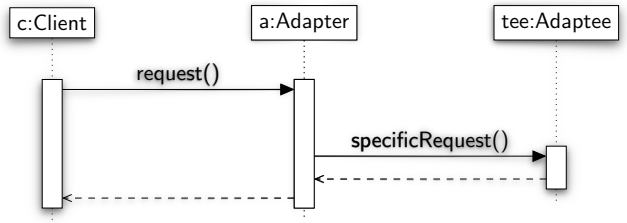
Design Pattern Adapter

- Soluzione Object Adapter
  - **Target** è l'interfaccia che il chiamante si aspetta
  - **Adaptee** è l'oggetto di libreria
  - **Adapter** converte, ovvero adatta, la chiamata che fa una classe client all'interfaccia della classe di libreria. Il chiamante usa l'Adapter come se fosse l'oggetto di libreria. Adapter tiene il riferimento all'oggetto di libreria (Adaptee) e sa come invocarlo, ovvero implementa le chiamate verso i metodi di Adaptee



Design Pattern Adapter

- Diagramma di sequenza della soluzione Object Adapter

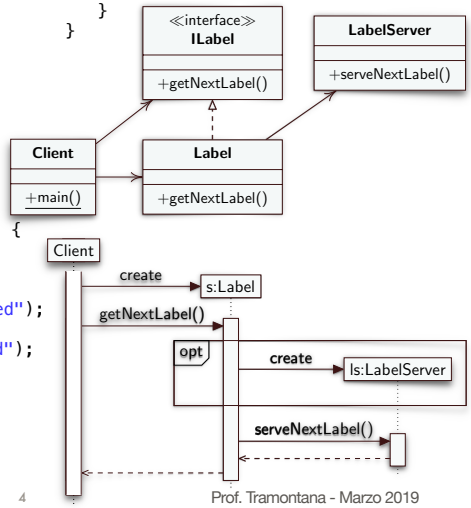



```
public interface ILabel { // Target
   public String getNextLabel();
}

// Adapter
public class Label implements ILabel {
   private LabelServer ls;
   private String p;
   public Label(String prefix) {
      p = prefix;
   }
   public String getNextLabel() {
      if (ls == null)
         ls = new LabelServer(p);
      return ls.serveNextLabel();
   }
}

public class Client {
   public static void main(String args[]) {
      ILabel s = new Label("LAB");
      String l = s.getNextLabel();
      if (l.equals("LAB1"))
         System.out.println("Test 1:Passed");
      else
         System.out.println("Test1:Failed");
   }
}
```

```
public class LabelServer { // Adaptee
   private int labelNum = 1;
   private String labelPrefix;
   public LabelServer(String prefix) {
      labelPrefix = prefix;
   }
   public String serveNextLabel() {
      return labelPrefix + labelNum++;
   }
}
```



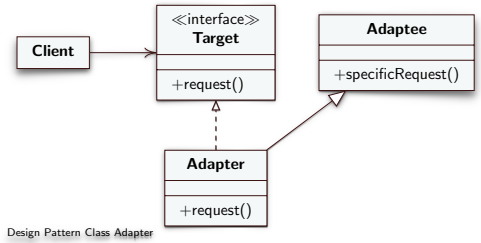
Design Pattern Adapter

- Soluzione Class Adapter
  - Adapter è sottoclasse di Adaptee

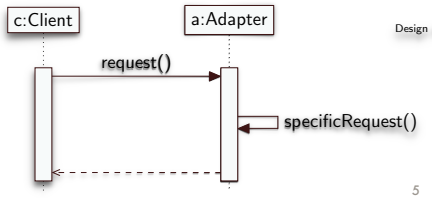
```
public class Label extends LabelServer implements ILabel { // Adapter

   public Label(String prefix) {
      super(prefix);
   }

   public String getNextLabel() {
      return serveNextLabel();
   }
}
```



Design Pattern Class Adapter



Design Pattern Adapter

- Variante Adapter a due vie
  - Definizione: la classe Adapter fornisce l'interfaccia di Target e l'interfaccia di Adaptee. Realizzazione: la soluzione Class Adapter è un Adapter a due vie
- Conseguenze del design pattern Adapter
  - Client e classe di libreria Adaptee rimangono indipendenti. L'Adapter può cambiare il comportamento dell'Adaptee
  - Può aggiungere test di precondizioni e postcondizioni [Precondizioni: cosa si deve soddisfare prima di eseguire. Postcondizioni: cosa è verificato se tutto è andato bene]
  - L'Object Adapter può implementare la tecnica di Lazy Initialization
  - Il design pattern Adapter aggiunge un livello di indirettezza. Ogni invocazione del client ne scatena un'altra fatta dall'Adapter. Possibile rallentamento (trascurabile), e codice più difficile da comprendere