Riportiamo qui il codice delle esercitazioni svolte durante il corso. Per mantenere tale codice allineato il più possibile con quanto visto a lezione, esso non è stato modificato in alcun modo rispetto all'ultimo editing svolto in classe. Le esercitazioni con numerazione progressiva rappresentano improvement apportati, rispetto ad una traccia originale, mettendo in pratica gli argomenti visti durante il corso.

Lo zip file si compone di 5 progetti Eclipse, riportiamo qui di seguito una breve descrizione degli stessi.

- **LMP**: Ereditarietà & Interfacce; Overloading & Overriding
  - Esercizio *Greeter*, dove si rappresenta l'uso di una semplice interfaccia (Greeter.java) con un singolo metodo hi(); si crea poi una implementazione della interfaccia (classe LMPGreeter.java), e si aggiungono dei metodi che sono invece specifici di tale classe. Creazione di una ulteriore sottoclasse di LMPGreeter.java (classe HelloLMPGreeter.java), che fa override di un metodo, cambiando l'implementazione rispetto alla sua superclasse.
  - Esercizio *Bike*, dove si definisce una classe che rappresenta biciclette. Si mostra l'utilità di metodi pubblici di modifica di variabili interne, al fine di poter monitorare i valori passati, e eventualmente verificare vincoli di consistenza. Creazione di una ulteriore classe (classe BikeWithConstructors.java) che offre diversi metodi costruttori per l'inizializzazione dei valori delle variabili interne, al momento della inizializzazione dell'oggetto.
- **Lmp1**: Abstraction & Encapsulation
  Prima fase dello sviluppo di un DB studenti. Si mostra come i metodi accessors&mutators permettano di offrire una interfaccia allo stato di un oggetto:
  - che astragga dalla rappresentazione interna di tale stato
  - che permetta di validare i valori proposti per cambiare lo stato

  Nello specifico, i valori passati per definire la matricola di un particolare tipo di studente (interi), non corrispondono allo stato interno della matricola (una stringa, garantita essere univoca attraverso l'intera collezione degli studenti, indipendentemente dal loro tipo).

- **Lmp2**: Classi Astratte e Pattern Factory ; enum
  - Utilizzo delle classi astratte, come livelli intermedi (tra interfacce e implementazioni) di codice che permettano di fattorizzare codice comune a più implementazioni. Nel caso specifico, StudenteImpl fornisce la implementazione, comune a tutte le sottoclassi di genere, del metodo di costruzione del numero di matricola.
  - La classe StudenteFactory fornisce un unico entry point che is occupa di creare studenti sapendo quali classi (implementazioni concrete) invocare. La scelta della classe da invocare non è quindi più delegata allo sviluppatore, che deve quindi passare un enum ad un metodo della factory per indicare quale tipologia di studente intende creare.

- **Lmp3**: Eccezioni: introduzione all'uso delle eccezioni nell'esercizio Studente.
- **Lmp4**: Java Reflection: La Factory ora crea automaticamente istanze delle classi opportune risolvendo il nome della classe tramite il metodo Class.forName(..).

Link prof progetti: course_classworks_2013-2014.zip
Link gDrive progetti: course classworks 2013-2014