**Prova in itinere N. 1 di Programmazione ad Oggetti**
**Corso di laurea in Ingegneria Informatica, a.a. 2016/17**
**Università degli Studi di Salerno**
**4 novembre 2016**

Si chiede di implementare l'insieme di classi ed interfacce elencate di seguito (alcune classi/interfacce sono già fornite):

- Interface
  - Filterable (già fornita)
  - Sortable (già fornita)
  - PersonFilter (già fornita)
- Enum
  - TeacherType (già fornita)
- Class
  - Person
  - Student
  - Teacher
  - Department
  - SelectStudentFilter
  - SelectYoungPersonFilter

Le suddette classi, interfacce ed enumerazioni[1] devono essere inserite tutte nel seguente package:

**oop2016.contest1.gruppoXX**

dove XX deve essere sostituito con il numero del proprio gruppo espresso su due cifre (ad es. il gruppo 4 userà il package oop2016.contest1.gruppo04).

Al candidato sono inoltre fornite le classi per effettuare i test del codice scritto ed in particolare:

- TestPerson
- TestStudent
- TestTeacher
- TestSelectYoungPersonFilter
- TestSelectStudentFilter
- TestDepartment

Le suddette classi sono già inserite nel seguente package:

**oop2016.contest1.test**

che non deve essere modificato.

[1] Per informazioni sulle enumerazioni consultare il seguente URL:
https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html

## Descrizione delle classi da implementare

Suggerimento: implementare le classi nell'ordine in cui sono presentate nel seguito usando le classi di test già fornite per assicurare la corretta implementazione del codice scritto prima di procedere all'implementazione delle classi successive.
Per ogni classe inoltre è specificato tra parentesi il punteggio attribuito in caso di corretto svolgimento. Il numero complessivo di classi da realizzare è pari a 6 il resto è già fornito.

### *Classe PERSON (PUNTEGGIO 8)*

- **[PUNTEGGIO 3]** - La classe *Person* definisce ed implementa l'entità persona caratterizzata dalle proprietà *name (String)*, *surname (String)*, e *birthDate (java.time.LocalDate)*.
  Le suddette proprietà sono in sola lettura e possono essere solo impostate al momento in cui viene istanziato l'oggetto mediante il seguente costruttore:
  *public Person(String name, String surname, int year, int month, int dayOfMonth)*
  senza che vi sia più la possibilità di modificarle.
  La classe rende disponibili i metodi di get di tutte le proprietà. Si ponga attenzione al metodo *getBirthDate()* affinchè esso ritorni una deep copy della data di nascita (ossia istanzi un nuovo oggetto *LocalDate* che contenga gli stessi dati della data di nascita dell'oggetto corrente) e non restituisca una shallow copy (quindi non ritorni una copia del riferimento della proprietà *birthDate*).
- **[PUNTEGGIO 2]** - La classe sovrascrive il metodo *toString()* di *Object* in modo da fornire a titolo di esempio in relazione ad una persona nata il 1 aprile 1998 di nome Matteo Baldi il seguente output:
  Name = Matteo
  Surname = Baldi
  Birth date = 1998-04-01
- **[PUNTEGGIO 3]** - La classe implementa il comportamento specificato dall'interface *Comparable<Persona>* definita dalle API di Java (*java.lang.Comparable<T>*); si consulti la documentazione online per verificare i metodi di tale interfaccia che bisogna implementare; in particolare in relazione al metodo *int compareTo(Person p)* il risultato fornito dal metodo deve essere tale per cui dati *Person p1, p2* allora *p1.compareTo(p2)* deve essere minore/maggiore di zero se *p1* è più giovane/anziano di *p2*, e uguale a 0 se *p1* e *p2* hanno la stessa data di nascita. Il metodo *compareTo* definisce la relazione d'ordine naturale sul tipo *Person*; tale relazione deve essere preservata per tutte le eventuali classi derivate da *Person* facendo il modo che il metodo non possa essere sovrascritto nelle classi derivate.

### *Classe STUDENT (PUNTEGGIO 4)*

- **[PUNTEGGIO 2]** - La classe *Student* definisce ed implementa l'entità studente che specializza l'entità persona aggiungendo la proprietà matricola (*String*).
  La nuova proprietà come quelle ereditate dalla superclasse è in sola lettura, non modificabile dopo la creazione dell'oggetto che può avvenire tramite il costruttore:
  *public Student(String name, String surname, String matricola, int year, int month, int dayOfMonth)*
- **[PUNTEGGIO 2]** - La classe sovrascrive il metodo *toString()* in modo da fornire, a titolo di esempio in riferimento ad uno studente nato il 15 febbraio 1994 di nome Marco Verdi con matricola 0612708762, il seguente output:
  Name = Marco
  Surname = Verdi

Birth date = 1994-02-15
Matricola = 0612708762

***Classe TEACHER (PUNTEGGIO 6)***

- **[PUNTEGGIO 4]** - La classe *Teacher* definisce ed implementa l'entità docente che specializza l'entità persona aggiungendo la proprietà *type (TeacherType)*; quest'ultima è un'enumerazione definita nel file *TeacherType.java* (già fornito). Una variabile di tale tipo può assumere solo tre valori *TeacherType.FULL_PROFESSOR*, *TeacherType.ASSOCIATE_PROFESSOR* e *TeacherType.LECTURER*
  La nuova proprietà, come quelle ereditate dalla superclasse, è in sola lettura, e non modificabile dopo la creazione dell'oggetto che può avvenire tramite il costruttore:
  *public Teacher(String name, String surname, TeacherType type, int year, int month, int dayOfMonth)*
- **[PUNTEGGIO 2]** - La classe sovrascrive il metodo *toString()* in modo da fornire, a titolo di esempio in riferimento ad un docente di tipo professore associato nato il 18 maggio 1973 di nome Gennaro Percannella, il seguente output:
  Name = Gennaro
  Surname = Percannella
  Birth date = 1973-05-18
  Type = ASSOCIATE_PROFESSOR

***Classe SELECTSTUDENTFILTER (PUNTEGGIO 2)***

- La classe *SelectStudentFilter* implementa l'interfaccia *PersonFilter* (già fornita) attraverso il metodo *boolean checkPerson(Person p)*. Una classe che implementa l'interfaccia *PersonFilter* definisce un proprio criterio per validare il parametro *p* restituendo *true* se una certa condizione è verificata, altrimenti *false* se la condizione non è verificata. Nello specifico, il metodo *checkPerson(p)* implementato dalla classe *SelectStudentFilter* restituisce *true* se *p* è un'istanza della classe *Student*, *false* altrimenti.

***Classe SelectYoungPersonFilter (PUNTEGGIO 2)***

- La classe *SelectYoungPersonFilter* implementa l'interfaccia *PersonFilter* (già fornita) attraverso il metodo *boolean checkPerson(Person p)*. Il metodo *checkPerson* implementato dalla classe *SelectYoungPersonFilter* restituisce *true* se la data di nascita di *p* è successiva alla data specificata attraverso il costruttore del filtro *SelectYoungPersonFilter*, false altrimenti. L'inizializzazione del filtro in oggetto è realizzata mediante l'invocazione del seguente costruttore:
  *public SelectYoungPersonFilter(int year, int month, int dayOfMonth)*

***Classe DEPARTMENT (PUNTEGGIO 10)***

- **[PUNTEGGIO 1] -** La classe *Department* definisce ed implementa l'entità dipartimento intesa come sequenza di *Person*. In particolare la classe specializza il tipo *LinkedList<Person>*.
- **[PUNTEGGIO 3] -** La classe *Department* implementa il comportamento *Sortable*. Il comportamento *Sortable* è specificato attraverso l'interface omonima (già fornita) che definisce il metodo *sort()*. Il metodo *sort()* consente di ordinare gli elementi contenuti nella classe che implementa *Sortable* in base alla relazione d'ordine naturale definito

dal tipo degli elementi stessi; nel caso di *Department* gli elementi in essa contenuti sono di tipo *Person* e la relazione d'ordine naturale è quella indotta dal metodo *compareTo()* di *Person* (crescente per età). Quindi dopo l'invocazione del metodo *sort()* su un oggetto di tipo *Department*, tutti gli elementi in *Department* dovranno essere ordinati in maniera crescente di data.
SUGGERIMENTO: per l'implementazione del metodo *sort()* verificare se la classe LinkedList dispone già di qualche metodo che può essere usato per effettuare l'ordinamento.

- **[PUNTEGGIO 4] –** La classe *Department* implementa anche il comportamento *Filterable*. Il comportamento *Filterable* è specificato attraverso l'interface omonima (già fornita) che definisce il metodo *List<Person> filter(PersonFilter f)*; il metodo *filter* implementato da *Department* restituisce un'oggetto istanza di una classe che implementa l'interfaccia *List<Person>* (ad esempio *LinkedList<Person>*, o anche la stessa *Department*) in cui sono presenti tutti e soli i riferimenti degli oggetti di tipo *Person* presenti in *Department* che soddisfano lo specifico criterio definito dall'istanza di *PersonFilter f* passata come parametro. Ad esempio dopo l'esecuzione del codice seguente:

  ```
  Department d;
  … inizializzazione di d mediante inserimento di persone, studenti e docenti…
  List<Person> d1 = d.filter( new SelectYoungPersonFilter(1995,1,1) );
  ```

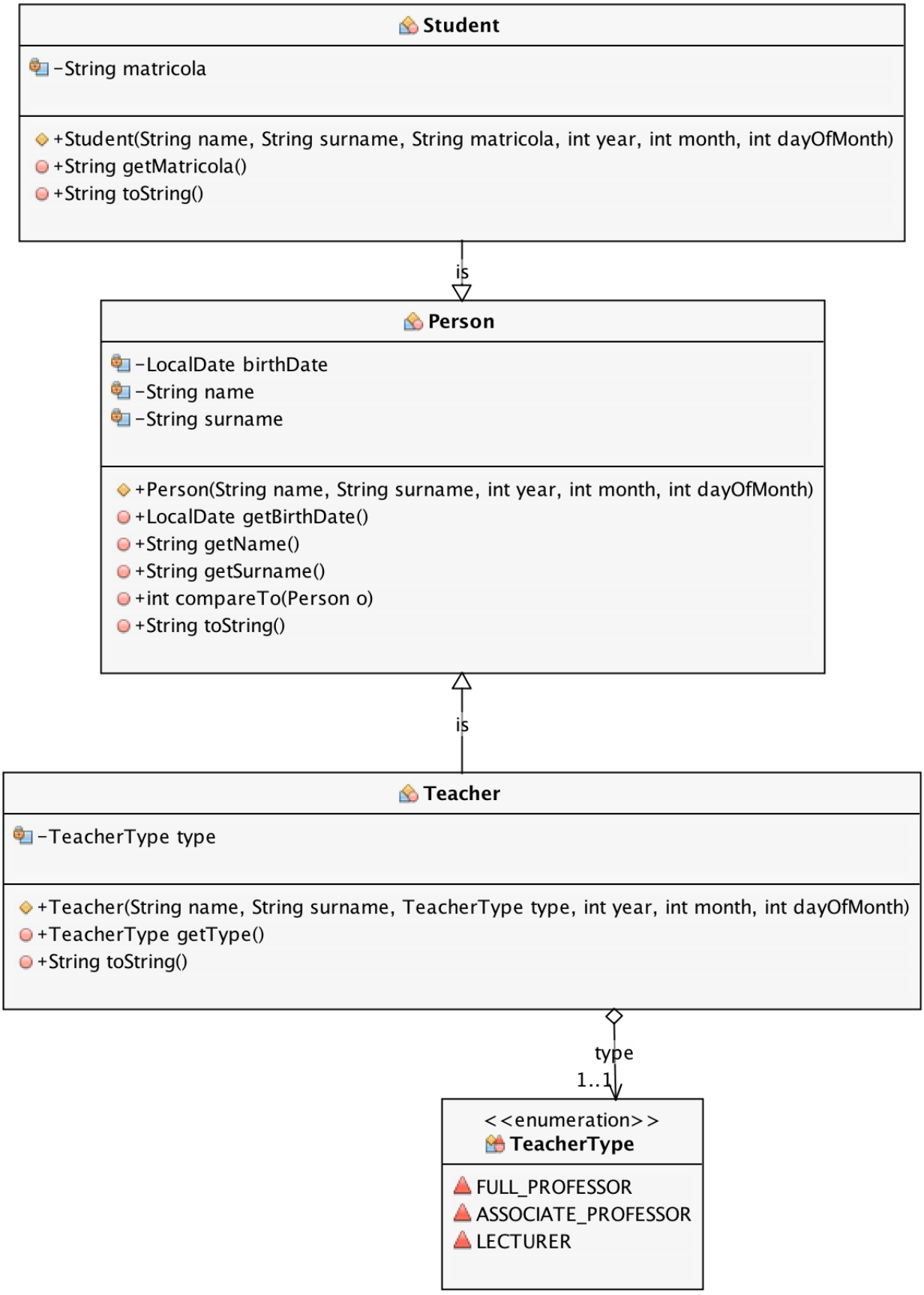
  la variabile *d1* conterrà i riferimenti di tutti e soli gli elementi di *d* la cui data di nascita sia successiva al 1 gennaio 1995; mentre invece dopo l'esecuzione del codice seguente:

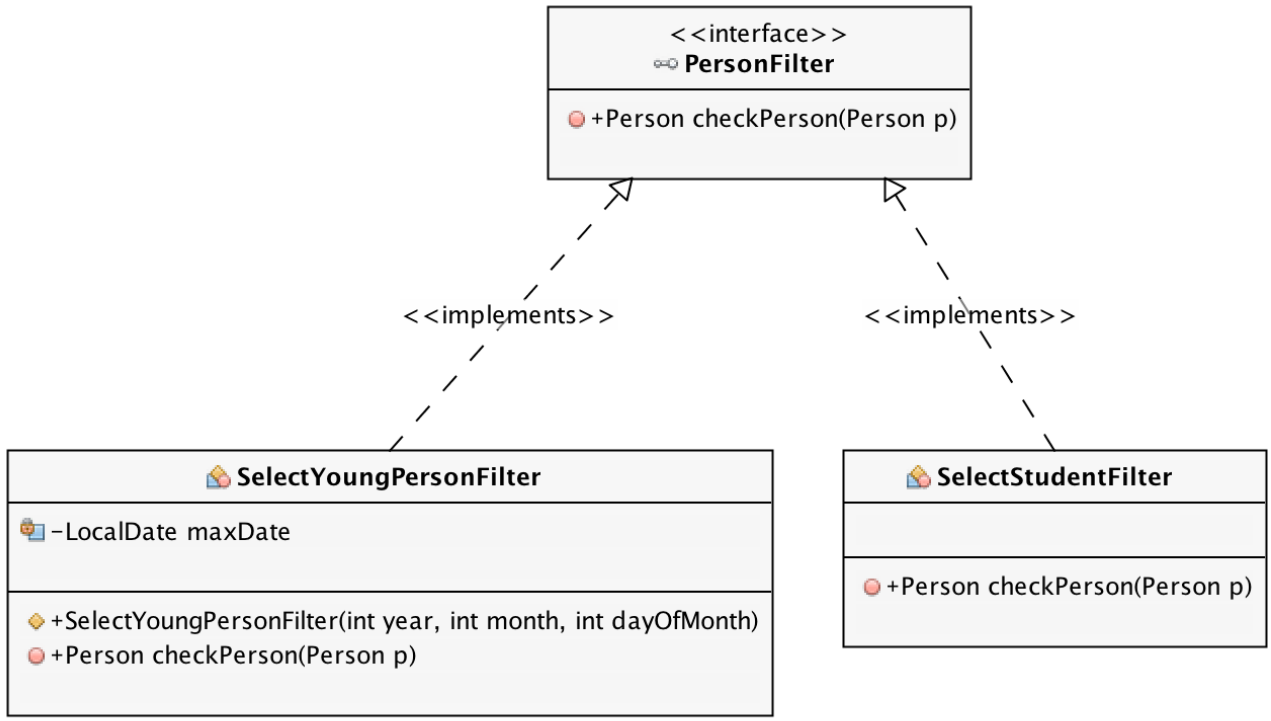
  ```
  List<Person> d2 = d.filter( new SelectStudentFilter() );
  ```

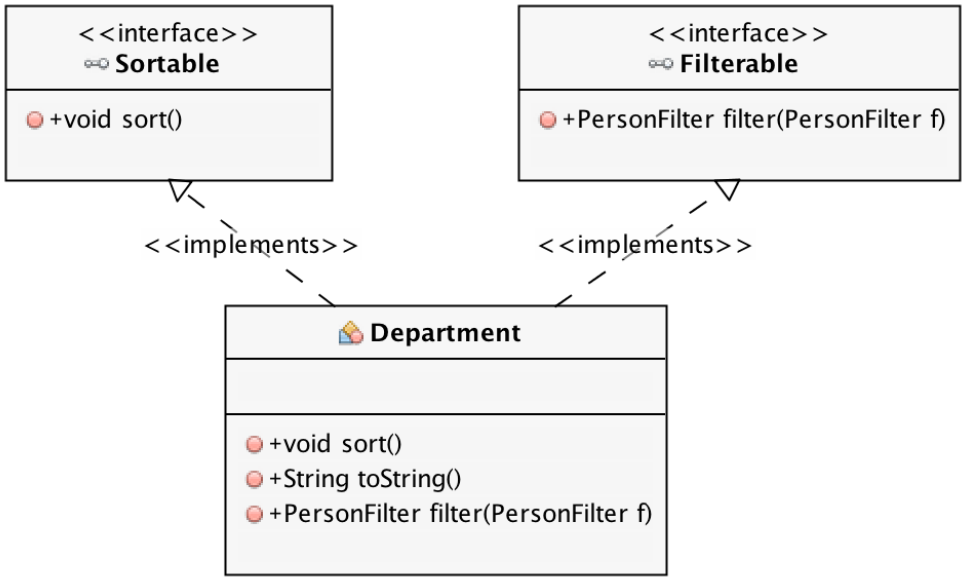
  la variabile *d2* conterrà i riferimenti di tutti e soli gli elementi di *d* istanze della classe Student. Si ponga attenzione al fatto che il metodo filter effettua una shallow copy (quindi copia i riferimenti nella nuova struttura dati e non duplica i singoli oggetti *Person*).
- **[PUNTEGGIO 2] –** La classe *Department* sovrascrive il metodo *toString()* in modo da fornire in output una stringa ottenuta concatenando tutti gli elementi presenti nell'istanza di *Department*; un esempio di output è riportato di seguito:

  ```
  Name = Luca
  Surname = Rossi
  Birth date = 1994-02-15
  Name = Luigi
  Surname = Bianchi
  Birth date = 1996-02-14
  Matricola = 0612701732
  Name = Gennaro
  Surname = Percannella
  Birth date = 1973-05-18
  Type = ASSOCIATE_PROFESSOR
  ```

Seguono i diagrammi delle classi da implementare.




<<interface>>
PersonFilter
+Person checkPerson(Person p)
<<implements>>
<<implements>>
SelectYoungPersonFilter
-LocalDate maxDate
+SelectYoungPersonFilter(int year, int month, int dayOfMonth)
+Person checkPerson(Person p)
SelectStudentFilter
+Person checkPerson(Person p)


<<interface>>
Sortable
+void sort()
<<interface>>
Filterable
+PersonFilter filter(PersonFilter f)
<<implements>>
<<implements>>
Department
+void sort()
+String toString()
+PersonFilter filter(PersonFilter f)