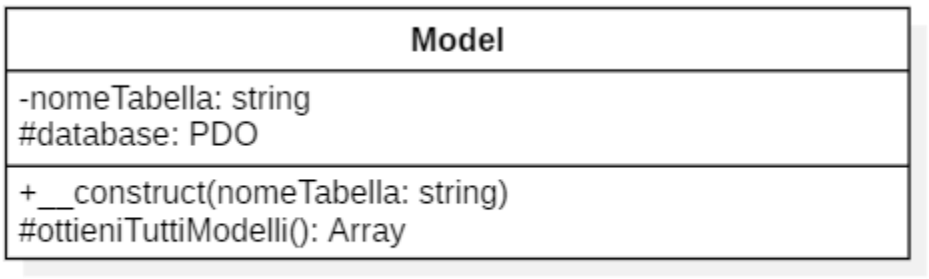
Diario di lavoro

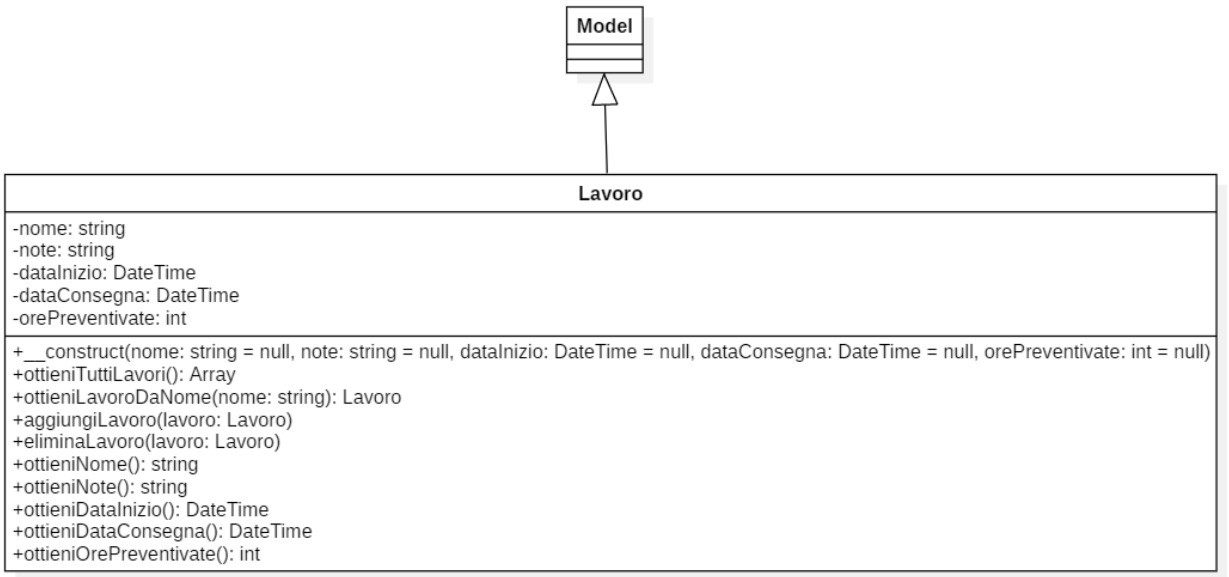
Titolo progetto	Gestione ore
Luogo	Trevano-Canobbio
Data	10 ottobre 2019

Lavori svolti

Durante la giornata di oggi mi sono occupato di implementare le classi Model che si interfacciano al database ed eseguono le query su di esso.
Più di preciso, ho cominciato aggiungendo una superclasse, "Model", che contenesse la connessione al database e il metodo per ottenere tutti i dati di una tabella, alla progettazione delle classi. Di seguito il suo schema UML:



Come è possibile vedere, questa classe contiene la connessione al database e un metodo generico "ottieniTuttiModelli" che ritorna un array associativo derivato da una query che ottiene tutti i dati della tabella corrispondente al campo nomeTabella del database.
Ho ovviamente dovuto modificare anche gli altri diagrammi, aggiungendo una generalizzazione verso questa superclasse, come è possibile vedere nell'immagine seguente, nella quale ho usato la classe Lavoro come esempio:



È importante notare come nello schema non abbia rimosso la funzione "ottieniTuttiLavori". Questo perché la funzione della superclasse ritorna un array associativo contenente il valore di tutti i campi della classe, mentre quello di questa sottoclasse e di tutte le altre ritorna un array del modello "corrente",

quindi la classe Lavoro ha il metodo “ottieniTuttiLavori” che prende il risultato del metodo “ottieniTuttiModelli” e lo usa per ritornare un array di oggetti di tipo Lavoro. Ho però dovuto impostare i metodi “ottieniTuttiLavori”, “ottieniLavoroDaNome”, “aggiungiLavoro” e “eliminaLavoro”, che prima erano statici, a non statici, perché altrimenti non è possibile utilizzare il valore del campo ‘database’, né la funzione “ottieniTuttiModelli”, entrambi ereditati dalla superclasse ‘Model’. È sempre per questo motivo che ho dovuto rendere opzionali tutti i parametri passati al metodo costruttore, per questa come per le altre classi: ora che i non possono più essere statici, le funzioni che si interfacciano al database devono poter essere chiamate anche senza assegnare dei valori ai campi ‘nome’, ‘note’, …

Dopo aver realizzato la classe Model sono potuto passare alle sottoclassi. Infatti oggi ho cominciato a implementare la classe model ‘Lavoro’, di cui sono riuscito a portare a termine la funzione “ottieniTuttiLavori”, che ho rinominato in “getAllActivities”, il cui codice è visibile di seguito:

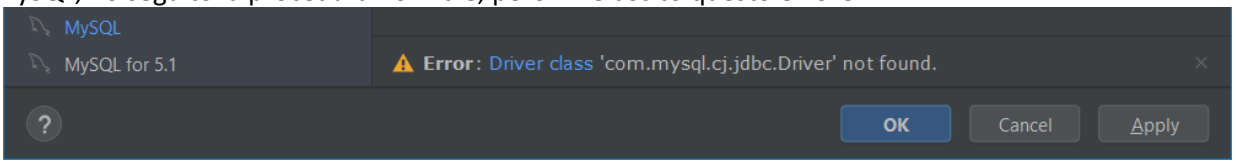
```
    /**
     * Ottiene tutti i lavori leggendoli dal database.
     * @return array Un array contenente un oggetto di tipo Lavoro per ogni riga letta dal database.
     */
    public function getAllActivities(): Array
    {
        //Istanzia l'array da ritornare
        $lavoriArray = [];
        //Ottieni i dati dal database grazie alla superclasse 'Model'
        $modelsArray = $this->getAllModels();
        //Cicla tutti gli elementi letti dal database e per ognuno aggiungi un elemento all'array lavoriArray
        //contenente un oggetto di tipo Lavoro con i dati dell'elemento corrente di modelsArray.
        foreach ($modelsArray as $model) {
            array_push(
                $lavoriArray,
                new Lavoro(
                    $model["nome"],
                    $model["note"],
                    DateTime::createFromFormat("Y-m-d", $model["data_inizio"]),
                    DateTime::createFromFormat("Y-m-d", $model["data_consegna"]),
                    intval($model["ore_preventivate"])
                )
            );
        }
        return $lavoriArray;
    }
```

Poi sono riuscito a realizzare anche la funzione “ottieniLavoroDaNome”, anche in questo caso rinominata in “getActivityByName”, sempre visibile in seguito:

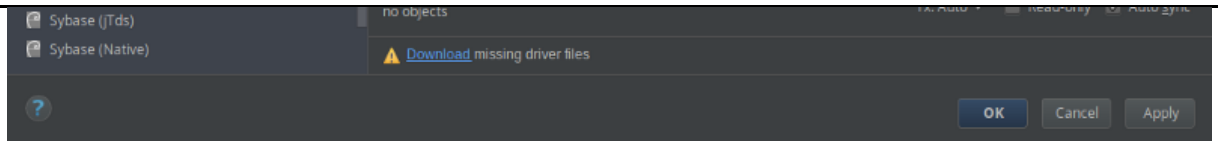
```php
    /**
     * Ottiene un lavoro con il nome specificato.
     * @param string $name Il nome per il quale cercare un lavoro all'interno del datab
ase.
     * @return Lavoro Un oggetto il cui valore dei campi è uguale ai dati
     * della riga il cui nome corrisponde al valore del parametro name.
     */
    public function getActivityByName(string $name): Lavoro
    {
        //Scrivi la query per la ricerca nel database
        $query = "SELECT * FROM lavoro WHERE nome = :name";
        //Prepara la query
        $statement = $this->database->prepare($query);
        //Assegna al segnaposto ':name' il valore del parametro name
        $statement->bindParam(":name", $name);
        //Esegui la query
        $statement->execute();
        //Vai a prendere i risultati
        $queryResult = $statement->fetchAll();
        //Se sono stati ritornati risultati
        if (count($queryResult) > 0) {
            //Ritorna un nuovo oggetto di tipo Lavoro con i dati del risultato
            return new Lavoro(
                $queryResult[0]["nome"],
                $queryResult[0]["note"],
                DateTime::createFromFormat("Y-m-d", $queryResult[0]["data_inizio"]),
                DateTime::createFromFormat("Y-m-d", $queryResult[0]["data_consegna"]),
                intval($queryResult[0]["ore_preventivate"])
            );
        }
        //Se arriviamo qui vuol dire che non è stato trovato alcun risultato, quindi ri
torna null
        return null;
    }
```

Problemi riscontrati e soluzioni adottate

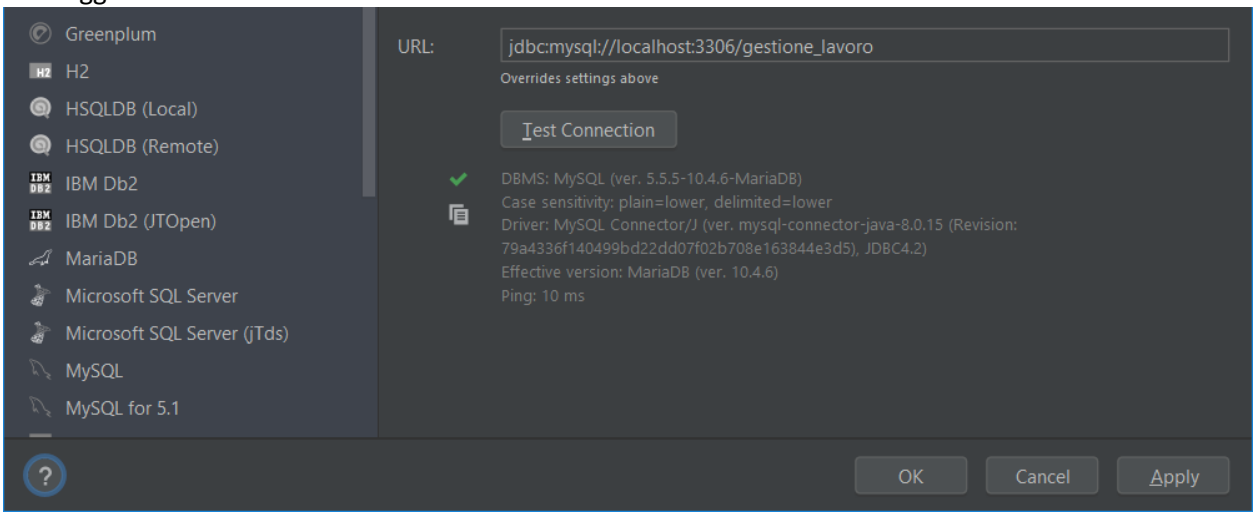
Quando ho tentato di configurare PHPStorm per prendere i dati per i suoi suggerimenti dal database MySQL, ho seguito la procedura normale, però mi è uscito questo errore:



Per risolvere questo problema mi è bastato scaricare la classe mancante tramite il link proposto dall’IDE cliccando sul seguente link:



Dopo poco la classe è stata scaricata e, cliccando sul pulsante “Test Connection” usciva il seguente messaggio di successo:



Più tardi ho avuto un problema con la creazione della classe Lavoro, sottoclasse di Model, perché nel suo metodo costruttore non riuscivo a chiamare quello della superclasse. Ho infine scoperto che bastava usare il seguente codice:

```
parent::__construct("lavoro");
```

Poi ho riscontrato problemi con il metodo “ottieniTuttiLavori” della classe model ‘Lavoro’: le date di inizio e di consegna vengono passate come stringhe e il metodo costruttore richiede un oggetto di tipo DateTime, questo vuol dire che dovevo trovare il modo di convertirle.
Ho poi scoperto che MySQL usa sempre lo stesso formato per le sue date, ovvero “YYYY-MM-DD”, quindi ho potuto usare il seguente codice:

```
DateTime::createFromFormat("Y-m-d", $model["data_inizio"])
```

Punto della situazione rispetto alla pianificazione

Sono in anticipo rispetto alla pianificazione.

Programma di massima per la prossima giornata di lavoro

Portare avanti la documentazione della pianificazione e la realizzazione delle classi Model che si interfacciano al database.