Diario di lavoro

Titolo progetto	Gestione ore
Luogo	Trevano-Canobbio
Data	22 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 terminato la funzione “isAssigned” della classe ‘Assignment’, per la quale mi è bastato sistemare un errore all’interno della funzione “equals” della classe ‘Activity’ (vedi sezione Problemi riscontrati e soluzioni adottate). Il codice, rispetto alla lezione scorsa, non è stato modificato.
Poi ho creato la funzione “addAssignment”, che è molto semplice perché ho potuto generalizzare il suo contenuto nella superclasse ‘Model’ (vedi sezione Problemi riscontrati e soluzioni adottate). Il codice che la compone è il seguente:

```
/**
 * Inserts a new row into table 'assegna' of database with data passed as parameter.
 * @param Assignment $assignment An object of type Assignment that contains the data to
 add to the database.
 * @return bool true if the insert operation is successful, false otherwise.
 */
public function addAssignment(Assignment $assignment): bool
{
    //Check if the assignment and its values (Activity and Resource) are not null, this
 includes all fields of both
    //the assignment's activity and resource.
    //Also, check if the values of the fields of both activity and resource are present
 in a single row of the database.
    if ($this->isValid()) {
        //Insert a new row into table 'assegna' using inherited function "addModel".
        return $this->addModel([$assignment->activity->getName(), $assignment-
>resource->getName()]);
    }
    return false;
}
```

Dopodiché ho implementato la funzione “deleteAssignment” che rimane semplice come le altre, sempre grazie alla funzione “deleteModel” ereditata dalla superclasse ‘Model’.
Il suo codice sorgente è il seguente:

```
/**
 * Deletes a record from the MySQL table 'risorsa' where the name is equal to the name
of an object of type Resource.
 * @param Resource $resource The data of the resource to delete.
 * @return bool true if the deletion is successful, false otherwise.
 */
public function deleteResource(Resource $resource): bool
{
    //If the resource is valid and its values are present in a single row of table 'ris
orsa' of the database, delete.
    if ($resource->isValid()) {
        if ($resource->equals($this->getResourceByName($resource->getName()))) {
            return $this->deleteModel([$resource->name]);
        }
    }
    return false;
}
```

Problemi riscontrati e soluzioni adottate

Il primo problema che ho trovato oggi è stato il fatto che utilizzassi la funzione “fetchAll” all’interno di “getModelByKey” quando in realtà a me serviva solamente un valore, per cui l’ho rimpiazzata con una chiamata alla funzione “fetch”:

```
//Fetch the results
$queryResult = $statement->fetch(PDO::FETCH_ASSOC);
```

In seguito ho notato che la funzione “equals” all’interno della classe ‘Activity’ restituiva sempre il valore `false`, questo perché per confrontare tra loro i valori dei campi di tipo DateTime, veniva usato l’operatore di identificazione, ossia `===`, mentre era necessario utilizzare l’operatore di uguaglianza `==`. Perciò adesso viene eseguito il seguente controllo:

```
$activity->startDate == $this->startDate &&
$activity->deliveryDate == $this->deliveryDate
```

Poi ho pensato che sarebbe stato più comodo generalizzare il controllo dei valori dei campi di un oggetto di tipo Activity e Resource in una funzione all’interno delle rispettive classi, per cui ho creato la funzione “isValid” all’interno di tutte le classi Model realizzate fino ad ora. Di per sé mi è bastato spostare in una funzione apposita la validazione che veniva fatta in cima alla funzione “add…”, che ho già documentato nel rapporto di lavoro di due giornate fa e quello della scorsa (vedi 2019.10.17.pdf e 2019.10.18.pdf).

Quella che segue è invece la funzione “isValid” della classe ‘Assignment’:

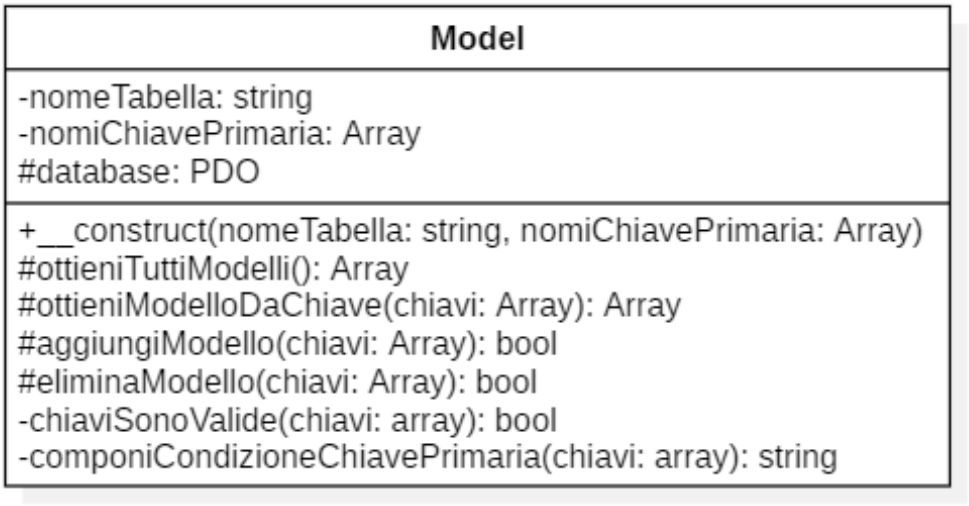
```
/**
 * Checks if the values of the fields of this object of type Assignment are valid.
 * The values are valid when this object is not null and the value that was returned fr
om the call to functions
 * isValid of this assignment's activity and resource is true. Also, the values of both
 fields have to be present in
 * a single row of their respective tables for this object to be valid.
 * @return bool true if the values of the fields of this object are valid, otherwise fa
lse.
 */
public function isValid()
{
    $databaseActivity = $this->activity->getActivityByName($this->activity->getName());
    $databaseResource = $this->resource->getResourceByName($this->resource->getName());

    return
        isset($this) &&
        $this->activity->isValid() &&
        $this->resource->isValid() &&
        $this->activity->equals($databaseActivity) &&
        $this->resource->equals($databaseResource);
}
```

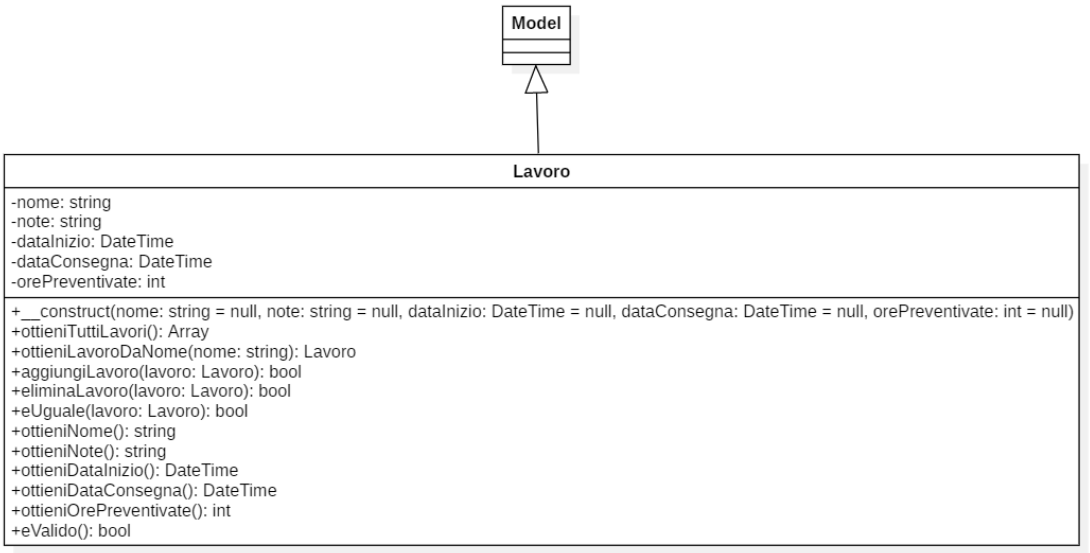
Più tardi mi sono reso conto del fatto che fosse possibile generalizzare pure le funzioni “add…” delle varie sottoclassi di ‘Model’ in modo da non doverle riscrivere ogni volta. Ho quindi creato la funzione “addModel”, che esegue una query generica per l’inserimento di una riga all’interno di una tabella del database. Il suo codice è visibile in basso:

```
/**
 * Inserts a model into the MySQL table whose name corresponds to the value of field ta
bleName.
 * @param array $keys An array that contains an element of type string for each primary
 key defined inside array primaryKeyNames.
 * @return bool true if the insertion was successful, false otherwise.
 */
protected function addModel(array $keys): bool
{
    //See if there's an invalid key in the array
    $keysValid = $this->areKeysValid($keys);
    if ($keysValid) {
        //Write the query that will write into the database
        $query = "INSERT INTO $this->tableName(" . implode(", ", $this-
>primaryKeyNames) . ") VALUES (";
        //Loop each element of array keys and create a placeholder for each of them
        for ($i = 0; $i < count($keys) - 1; ++$i) {
            $query .= ":key$i, ";
        }
        $query .= ":key" . count($keys) - 1 . ")";
        //Prepare the statement
        $statement = $this->database->prepare($query);
        //Bind each placeholder to its respective key
        for ($i = 0; $i < count($keys); ++$i) {
            $statement->bindParam(":key$i", $keys[$i]);
        }
        //Execute the statement
        return $statement->execute();
    }
    return false;
}
```

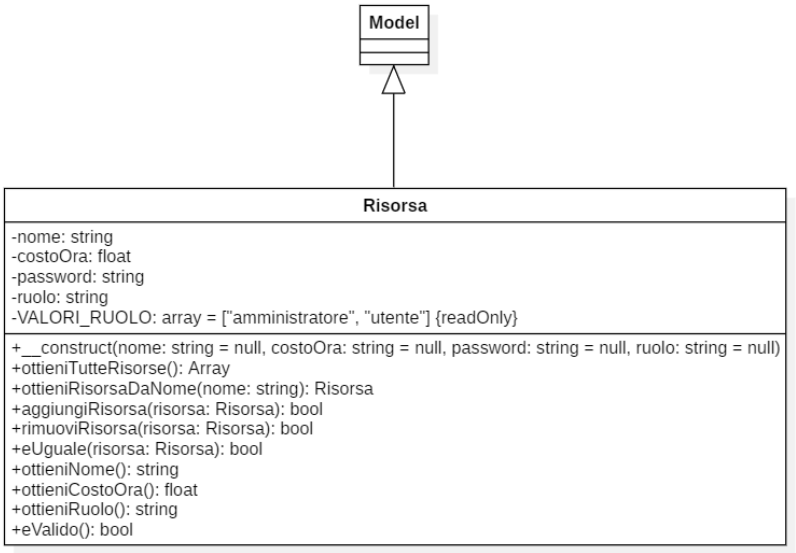
Per riassumere i metodi aggiunti, di seguito tutti i diagrammi UML aggiornati delle classi model realizzate finora:



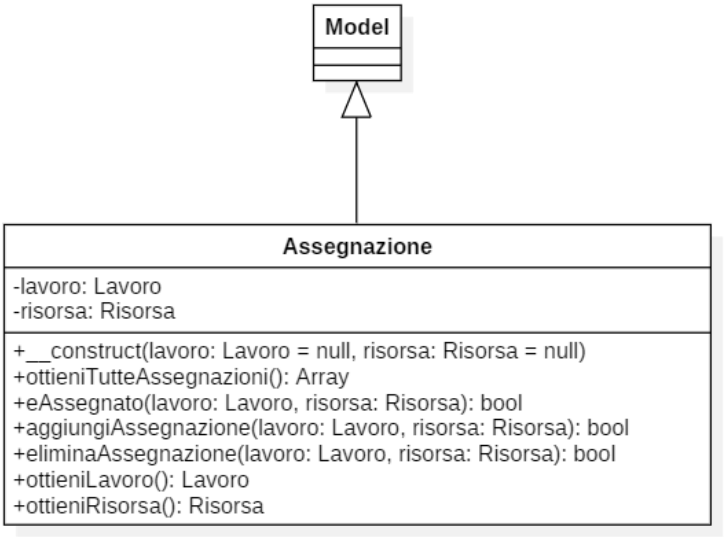
La superclasse ‘Model’, dove è stata aggiunta la funzione “aggiungiModello”.



Quella sopra è la classe ‘Lavoro’, le cui funzioni “eUguale” e “eValido” sono nuove.



Questa precedente è la classe 'Risorsa', dove ho creato le funzioni "eUguale" e "eValido".



Questa è la classe 'Assegnazione', nella quale non è stato aggiunto né tolto nulla.

Dopodiché ho constatato che non veniva eseguito alcun controllo prima dell'eliminazione di nessuna delle classi Model, per cui ho dovuto aggiungerli.
Nella classe 'Activity':

```
//If the activity is valid and its values are present in a single row of table 'lavoro'
 of the database, delete.
if ($activity->isValid()) {
    if ($activity->equals($this->getActivityByName($activity->getName()))) {
        return $this->deleteModel([$activity->name]);
    }
}
return false;
```

Nella classe 'Resource':

```
//If the resource is valid and its values are present in a single row of table 'risorsa
' of the database, delete.
if ($resource->isValid()) {
    if ($resource->equals($this->getResourceByName($resource->getName()))) {
        return $this->deleteModel([$resource->name]);
    }
}
return false;
```

Nella classe 'Assignment':

```
//If the assignment is valid and its values are present in a single row of table 'asseg
na' of the database, delete.
if ($assignment->isValid()) {
    return $this->deleteModel([$assignment->activity, $assignment->resource]);
}
return false;
```

Poi ho visto che l'algoritmo che veniva utilizzato per criptare la password di una risorsa prima del suo inserimento nel database sarebbe potuto cambiare se in futuro ci fossa stata una modifica dell'algoritmo predefinito in seguito a una nuova versione di PHP. Ho quindi modificato l'algoritmo utilizzato:

```
$securePassword = password_hash($resource->password, PASSWORD_BCRYPT);
```

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.