Petshop

Tenche Eduard & Tuslea Andreea

Class Reference:

- Annotations
  - AutoIncrement
  - CLICommand
  - PrimaryKey
- Base
  - CacheData
  - Database
  - DatabaseProxy
  - DatabaseStrategy
  - Observable
  - Observer
  - Repo
  - Table
- Repositories
  - ListRepo
- Strategies
  - ListSQLAdapter
  - ListStrategy
  - MySQLStrategy
- Helpers
  - ArrayHelper
  - StringConverter
- Factories
  - ProxyFactory
- Controllers
  - Controller
  - AnimalController
  - AnimalTypeController
  - ClientController

    - HabitatController
    - ReceiptController
    - ReceiptAnimalController
- Tables
    - Animal
    - AnimalType
    - Client
    - Habitat
    - Receipt
    - ReceiptAnimal
- UI
    - CLI
    - REST

AutoIncrement

Description

An Annotation used on Primary Key fields of Type Integer in classes that extend Table. When an object that has an attribute with this annotation is added to a Repo, it first searches for the highest existing value for the attribute in the repo, and increments it by 1 for the newly added object.

@Target is ElementType.FIELD

@Retention is RetentionPolicy.RUNTIME

CLICommand

Description

An Annotation used in the CLI class to mark a method as being available to be called by the user in the CLI. A CLICommand should return a String, and it should have a String[] as its only parameter.

@Target is ElementType.METHOD

@Retention is RetentionPolicy.RUNTIME

PrimaryKey

Description

An Annotation used on Fields in classes extending Table. It marks that Field as being unique, and is then used when comparing objects (when finding, adding, etc). Objects of the same type with the same values for its Primary Keys are considered equal, regardless of other attributes.

@Target is ElementType.FIELD

@Retention is RetentionPolicy.RUNTIME

CacheData

Description

A class that is used to hold information about a cached user command.

Properties:

String	table
String	function
String[]	parameters
String	output

Constructors:

CacheData (String table, String function, String[] parameters, String output)

Database extends Observable implements Observer

Description

Represents a Database. It can use different Strategies to determine how the data is stored and used.

Properties:

String	name
DatabaseStrategy	strategy

Constructors:

Database (**String** name)
Database (**String** name, **DatabaseStrategy**strategy)

Methods:

DatabaseStrategy	getStrategy ( )
void	setStrategy (**DatabaseStrategy** strategy)
boolean	add (**Table** table) *throws Exception*
boolean	remove (**Table** table) *throws Exception*
Table[]	getTables ( )
Table []	getTables (**String** tableName)
String	getName ( )
void @Override	update (**Object** arg)

DatabaseStrategy getStrategy ( )

- Returns the *strategy* property

void setStrategy (**DatabaseStrategy** strategy)

- Sets the *strategy* property

boolean add (**Table** table) *throws Exception*

- Adds a table to the database
- Returns true if table was added successfully
- Returns false otherwise
- Notifies all proxies in order to clear cache that uses this table

boolean remove (**Table** table) *throws Exception*

- Removes a table from the database
- Returns true if table was removed successfully
- Returns false otherwise
- Notifies all proxies in order to clear cache that uses this table

Table[] getTables ( )

- Returns all the tables in the database

Table[] getTables (**String** tableName)

- Returns all the tables of a certain type in the database

String getName ( )

- Returns the *name* property

void update (**Object** arg)

- Object can be of type **String** or **DatabaseProxy**
- Receiving an update with an object of type **String** will send back the result of *getTables(arg)*.
- Receiving an update with an object of type **DatabaseProxy** will add that Proxy to the Observer list

DatabaseProxy extends Observable implements Observer

Description

Makes the connection between the User and the Database. All user Inputs are sent to the proxy, and the proxy then retrieves the necessary information from the database and sends it back to the user.

Properties:

CacheData[]	cachedData
Database	database

Constructors:

DatabaseProxy ( )
DatabaseProxy (**Database** database)

Methods:

String	getDatabaseName ( )
DatabaseStrategy	getStrategy ( )
String	getTablesByName (**String** tableName)
void	addTable (**String** tableName) *throws Exception*
void	addTable (**String** tableName, **String…** args) throws Exception
void	removeTable (**String** tableName, **String…** args) *throws Exception*
CacheData	getCachedData (**String** tableName, **String** function, **String[]** parameters)
void	cache (**String** tableName, **String** function, **String[]** parameters, **String** output)
void @Override	update (**Object** arg)

String getDatabaseName ( )

- Returns *database.name*

DatabaseStrategy getStrategy ( )

- Returns *database.strategy*

String getTablesByName (**String** tableName)

- Returns tables of type specified in *tableName*

void addTable (**String** tableName) *throws Exception*

- Creates a new table of type specified in *tableName* and adds it to the database.

void addTable (**String** tableName, **String…** args) *throws Exception*

- Creates a new table of type specified in *tableName* with its properties defined in *args.*
- *args* can either be of format "attributeName=value" or just "value", but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

void removeTable(**String** tableName, **String…** args) *throws Exception*

- Removes all tables of type specified in tableName which meet the conditions set in *args.*
- *args* can either be of format "attributeName=value" or just "value", but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

CacheData getCachedData(**String** tableName, **String** function, **String[]** parameters)

- Returns the cached data for the specified table, function, and parameters or null if it doesn't exist.

void cache (**String** tableName, **String** function, **String[]** parameters, **String** output)

- Caches a user-inputted command and its output.

void update (**Object** arg)

- If arg is of class Table, looks for all data cached with that table and removes it.
- This update is received from a Database, when a table of that type is added or removed, meaning that on next query the output has to be calculated again

DatabaseStrategy

Description

An interface that declares all the necessary methods for a database strategy.

Methods:

boolean	add (**Table** table) *throws Exception*
boolean	remove (**Table** table) *throws Exception*
Table[]	get (**String** name)
Table[]	getAll ( )

Observable

Description

An abstract class that declares and defines all the necessary properties and methods for an Observable object.

Properties:

Observer[]	observers

Methods:

void	notifyObservers ( )
void	notifyObservers (**Object** arg)
void	notifyObservers (**Object[]** args)

void notifyObservers ( )

- Calls update method of all Observers in *observers* with no arguments

void notifyObservers (**Object** arg)

- Calls update method of all Observers in *observers* with one argument

void notifyObservers (**Object[]** args)

- Calls update method of all Observers in *observers* with multiple arguments

Observer

Description

An interface that declares all the necessary methods for an Observer object.

Methods:

void	update ( )
void	update (**Object** arg)
void	update (**Object[]** args)

void update ( )

- Called whenever an **Observable** that is observed by this object calls its *notifyObservers* method with no arguments.

void update (**Object** arg)

- Called whenever an **Observable** that is observed by this object calls its *notifyObservers* method with one arguments.

void update (**Object[]** args)

- Called whenever an **Observable** that is observed by this object calls its *notifyObservers* method with multiple arguments.

Repo<T>

Description

An abstract class that declares and defines all the necessary properties and methods for a Repository that holds tables.

Properties:

Class<T>	type

Constructors:

protected **Repo** ( )
Repo (**Class<T>** type)

Methods:

boolean	add (**Object** obj) *throws Exception*
boolean	addToRepo (**T** instance)
boolean	remove (**Object** obj)
boolean	removeFromRepo(**T** instance)
T	getValue (**int** index)
Integer	getMaxIncrement (**Field** field) *throws Exception*
int	getSize ( )
boolean	contains (**Object** obj)
Class<?>	getType ( )
T	castToGeneric (**Object** obj)

boolean add (**Object** obj)

- Casts *obj* to type **T** if possible and adds it to the Repo using *addToRepo*.
- This method also runs all the @AutoIncrement logic, so adding tables to the repo directly using *addToRepo* will not automatically increment any attributes that have that annotation.

boolean addToRepo (**T** instance)

- Adds *instance* to the Repo and returns true if successful.

boolean remove(**Object** obj)

- Casts *obj* to type **T** if possible and removes it from the Repo using *removeFromRepo if it contains it*.

boolean removeFromRepo(**T** instance)

- Removes *instance* from the Repo and returns true if successful.

T getValue (**int** index)

- Gets the value of the object at *index*.

Integer getMaxIncrement (**Field** field)

- Returns the highest value of the specified field in the Repo.

int getSize ( )

- Returns the amount of tables in the Repo.

boolean contains (**Object** obj)

- Returns true if *obj* is in the Repo.

Class<?> getType ( )

- Returns the *type* property.

T castToGeneric(**Object** obj)

- Casts *obj* to type **T** or null if it cannot be casted. Used mostly internally.

Table

Description

An abstract class that declares all the methods and properties necessary for a table object. Objects extending Table must have an attribute with the **PrimaryKey** annotation, or else they will throw an exception whenever instantiated. Objects extending Table should have lowercase names and attribute names (because mysql lowercase shit).

Properties:

Field[]	primaryKeys

Constructors:

Table ( ) *throws Exception*

Methods:

Field[]	getFields ( )
Field	getField (**Integer** index)
Field	getField (**String** name)
Field[]	getPrimaryKeys ( )
Object	getProperty (**Integer** index) *throws IllegalAccessException*
Object	getProperty (**String** name) *throws IllegalAccessException*
void	setProperty (**Integer** index, **Object** value) *throws Exception*
void	setProperty (**String** name, **Object** value) *throws Exception*
boolean	checkPrimaryKey ( ) *throws Exception*

Field[] getFields ( )

- Returns all properties of the table.

Field getField (**Integer** index)

- Returns the field at *index*. Index is based on the order in which the Fields are declared in the class.

Field getField (**String** name)

- Returns the field by *name*.

Field[] getPrimaryKeys ( )

- Returns all fields with the **PrimaryKey** annotation.

Object getProperty(**Integer** index) *throws IllegalAccessException*

- Returns the value of the **Field** at *index.* Index is based on the order in which the Fields are declared in the class.

Object getProperty(**String** name) *throws IllegalAccessException*

- Returns the value of the **Field** by *name.*

Object setProperty(**Integer** index, **Object** value) *throws Exception*

- Finds **Field** at *index* and changes its value to *value.* Index is based on the order in which the Fields are declared in the class.

Object setProperty(**String** name, **Object** value) *throws Exception*

- Finds **Field** by *name* and changes its value to *value*.

boolean checkPrimaryKey ( ) *throws Exception*

- Returns false if the table has no properties with the **PrimaryKey** annotation.

ListRepo<T> extends Repo<T>

Description

An implementation of Repo<T> using ArrayLists.

Properties:

List<T>	list

Constructors:

protected **ListRepo** ( )

Methods:

static **ListRepo<S>**	newRepo (**Class<S>** clazz) *throws Exception*

static **ListRepo<S>** newRepo (**Class<S>** clazz) *throws Exception*

- Returns a new Repo of type S.

ListSQLAdapter extends ListStrategy

Description

An adapter class that converts a MySQLStrategy into a ListStrategy.

Properties:

MySQLStrategy	sqlStrategy

Constructors:

ListSQLAdapter (**MySQLStrategy** sqlStrategy) *throws Exception*

ListStrategy implements DatabaseStrategy

Description

An implementation of DatabaseStrategy using an ArrayList.

Properties:

List<ListRepo<?>>	list

Methods:

boolean	contains (**Repo<?>** repo)

boolean contains (**Repo<?>** repo)

- Returns true if *list* contains *repo.*

MySQLStrategy implements DatabaseStrategy

Description

An implementation of DatabaseStrategy using a connection to a MySQL server.

Properties:

Connection	connection
String	dbName

Constructors:

MySQLStrategy (**String** address, **String** port, **String** dbName, **String** username, **String** password)

Methods:

Table[]	getTablesFromQuery (**String** query)
String	query (**String** query)

Table[] getTablesFromQuery (**String** query)

- Returns an SQL query as an array of **Table** objects.

String query (**String** query)

- Returns an SQL query formatted as a string.

ArrayHelper

Description

A helper class that contains static methods to make working with arrays easier.

S is a generic type declared in every method as <S extends Object>

Methods:

S[]	addElement (**S[]** array, **S** object, **int** index)
S[]	removeElement (**S[]** array, **int** index)
S[]	addElement (**S[]** array, **S** object)
boolean	contains (**S[]** array, **S** object)
S[]	clone (**S[]** array, **int** startFrom, **int** EndAt)
S[]	clone (**S[]** array, **int** startFrom)
S[]	except (**S[]** array, **S[]** array2)
S[]	concatenate (**S[]** array, **S[]** array2)

S[] addElement (**S[]** array, **S** object, **int** index)

- Returns a new array that is a copy of *array* with *object* at *index.*

S[] removeElement (**S[]** array, **int** index)

- Returns a new array that is a copy of *array* without the object at *index.*

S[] addElement (**S[]** array, **S** object)

- Returns a new array that is a copy of *array* with *object* at the end.

boolean contains(**S[]** array, **S** object)

- Returns true if *array* contains *object.*

S[] clone (**S[]** array, **int** startFrom, **int** EndAt)

- Returns a clone of *array* that starts at *startFrom* and ends at *EndAt*.

S[] clone (**S[]** array, **int** startFrom)

- Returns a clone of *array* that starts at *startFrom.*

S[] except (**S[]** array, **S[]** array2)

- Returns an array that has all elements of *array* that are not in *array2*.

S[] concatenate (**S[]** array, **S[]** array2)

- Returns an array that is a clone of *array* with *array2* concatenated at the end.

StringConverter

Description

A helper class that contains a static method to convert different strings to objects. Used mainly when interpreting user input.

Methods:

Object	convert(**String** string, **Class<?>** converted) *throws Exception*

Object convert(**String** string, **Class<?>** converted)

- Converts *string* to an **Object** of class *converted.*

ProxyFactory

Description

A factory class that creates **DatabaseProxies**.

Methods:

DatabaseProxy	create(**String** name, **String** type, **String…** args)
String[]	options ( )

DatabaseProxy create(**String** name, **String** type, **String…** args)

- Creates a new **DatabaseProxy** that contains a **Database** with *name,* a strategy matching *type*. Args is used for further instructions, and are necessary for a **MySQLStrategy** in order to connect to the MySQL Server.

Controller

Description

A controller class which handles all the commands possible to be run by user input.

Properties:

DatabaseProxy[]	proxies

Methods:

DatabaseProxy	getProxy (**String** name)
String	getCachedOutput (**DatabaseProxy** proxy, **String** table, **String** function, **String[]** parameters)
void	cache (**DatabaseProxy** proxy, **String** table, **String** function, **String[]** parameters, **String** output)
String	execQuery (**String** dbName, **String** query)
String	createDatabase (**String** name, **String** choice, **String…** extraArgs)
String	table (**String** dbName, **String** tableName)
String	addTable (**String** dbName, **String** tableName, **String…** extraArgs) *throws Exception*
String	removeTable (**String** dbName, **String** tableName, **String…** extraArgs) *throws Exception*
String	hasTable (**String** dbName, **String** tableName)

DatabaseProxy getProxy(**String** name)

- Gets a proxy whose *database.name* is equal to *name.*

String getCachedOutput(DatabaseProxy proxy, **String** table, **String** function, **String[]** parameters)

- Returns the output from a CacheData matching the parameters.

void cache (DatabaseProxy proxy, **String** table, **String** function, **String[]** parameters, **String** output)

- Creates a CacheData object in *proxy* matching the arguments.

String execQuery (**String** dbName, **String** query)

- Returns the result of an SQL query (only works on a MySQLStrategy)

String createDatabase(**String** name, **String** choice, **String…** extraArgs)

- Creates a new DatabaseProxy of type matching *choice***.**

String table (**String** dbName, **String** tableName)

- Returns a list of all the tables of type matching *tableName* from the Database with name *dbName.*

String addTable (**String** dbName, **String** tableName, **String…** extraArgs) *throws Exception*

- Adds a table of type *tableName* to the Database with name *dbName***.**
- *extraArgs* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

String removeTable (**String** dbName, **String** tableName, **String…** extraArgs) *throws Exception*

- Removes all **tables** of type *tableName* matching the conditions set in *extraArgs* to the **Database** with name *dbName***.**
- *extraArgs* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

boolean hasTable (**String** dbName, **String** tableName)

- Returns true if the **Database** with name *dbName* contains **tables** of type *tableName.*

AnimalController

Description

A co

Methods:

String	addAnimal (**Controller** controller, **String** dbName, **Integer** idAnimal, **String** name, **Date** date, **Integer** animalType) *throws Exception*
String	removeAnimal (**Controller** controller, **String** dbName, **String...** conditions) *throws Exception*
String	getAnimals (**Controller** controller, **String** dbName)

String addAnimal (**Controller** controller, **String** dbName, **Integer** idAnimal, **String** name, **Date** date, **Integer** animalType) *throws Exception*

- Creates an **Animal** object in the **Database** with name *dbName* from *controller.*

String removeAnimal (**Controller** controller, **String** dbName, **String...** conditions) *throws Exception*

- Removes all **Animal** objects matching *conditions* in the **Database** with name *dbName* from *controller.*
- *conditions* can either be of format “attributeName=value” or just “value”, but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

String getAnimals (**Controller** controller, **String** dbName)

- Returns a list of all **Animal** objects in the **Database** with name *dbName* from *controller.*

animal extends Table

Description

A table representing an animal.

Properties:

@PrimaryKey Integer	idAnimal.
String	name
Date	date
Integer	animalType

animaltype extends Table

Description

A table representing an animaltype.

Properties:

@PrimaryKey Integer	idType
String	name

client extends Table

Description

A table representing a client.

Properties:

String	name
String	lastName
String	email
@PrimaryKey Integer	idClient

habitat extends Table

Description

A table representing a habitat.

Properties:

@PrimaryKey Integer	idHabitat
Integer	idType

receipt extends Table

Description

A table representing a receipt.

Properties:

Integer	idClient
@PrimaryKey Integer	idReceipt
Date	date

receiptanimal extends Table

Description

A table creating the many-to-many connection between receipt and animal.

Properties:

@PrimaryKey Integer	idReceipt
@PrimaryKey Integer	idAnimal
Integer	price

CLI

Description

A Command Line Interface to allow the user to interact with the database.

Properties:

Controller	controller
boolean	stop
int	exitCode
boolean	printStackTrace

Constructors:

CLI (**Controller** controller)

Methods:

void	invoke (**String** input)
void	start ( )
@CLICommand String	tgs (**String[]** args)
@CLICommand String	exit (**String[]** args)
@CLICommand String	toggleStackTrace (**String[]** args)
@CLICommand String	execQuery (**String[]** args)
@CLICommand String	createDatabase (**String[]** args)
@CLICommand String	cdb (**String[]** args)
@CLICommand String	table (**String[]** args)
@CLICommand String	addTable (**String[]** args) *throws Exception*
@CLICommand String	removeTable (**String[]** args) *throws Exception*
@CLICommand String	hasTable (**String[]** args) *throws Exception*

boolean stop

- Determines whether the while loop from *start ( )* should end.

int exitCode

- Exit code used to determine the state that the CLI exited in.

boolean printStackTrace

- If true, prints entire stack trace when an exception is thrown from an invoked command.
- If false, prints only “Could not invoke command”.

void invoke (**String** input)

- Invokes a command annotated with **@CLICommand**.
- Syntax: “*commandname arg1 arg2 arg3 arg4…”*
- Splits *input* into multiple strings using the format: [“arg1”, “arg2”, “arg3”, “arg4”…]

void start ( )

- As long as *stop* is false, takes user input and calls *invoke* with it.

@CLICommand String tgs (**String[]** args)

- Short form for *toggleStackTrace ( ).*

@CLICommand String exit (**String[]** args)

- Exits the CLI.
- *args*[0] is the exit code.

@CLICommand String toggleStackTrace (**String[]** args)

- If no argument is given, toggles *printStackTrace*
- *args[0]* can be “1” or “True” in order to set it to true, or “0” or “False” in order to set it to false.

@CLICommand String execQuery (**String[]** args)

- Executes a MySQL query using *controller.execQuery ( ).*
- Arguments: [*dbName, query*]

@CLICommand String createDatabase (**String[]** args)

- Creates a **Database** using *controller.createDatabase ( ).*
- Arguments: [*name, strategy, args*]

@CLICommand String cdb (**String[]** args)

- Short form for *createDatabase ( ).*

@CLICommand String table (**String[]** args)

- Returns a list of all tables with the specified name using *controller.table( )*.
- Arguments: [*dbName, tableName*]

@CLICommand String addTable (**String[]** args)

- Adds a table to the specified database with the given arguments using *controller.addTable ( ).*
- Arguments: [*dbName, tableName, extraArgs*]

@CLICommand String removeTable (**String[]** args)

- Removes all tables from the specified database with the given arguments using *controller.removeTable ( ).*
- Arguments: [*dbName, tableName, extraArgs*]

@CLICommand String hasTable (**String[]** args)

- Returns the output of *controller.hasTable ( )* using the given arguments.
- Arguments: [*dbName, tableName*]

REST

Description

A **RestController** that uses Spring Boot in order to allow the user to interact with the database using Posts.

Properties:

Controller	controller
String	password
AnimalController	animalController
AnimalTypeController	animalTypeController
ClientController	clientController
HabitatController	habitatController
ReceiptAnimalController	receiptAnimalController
ReceiptController	receiptController

Constructors:

Rest ( )

Methods:

void	setController (**Controller** controller)
@GetMapping("/print") String	print (**String** message)
@GetMapping("/query") String	query (**String** dbName, **String** query)
@GetMapping("/add[TableName]") String	add[TableName] (**String** dbName, [*attributes*])
@GetMapping("/remove[TableName]") String	remove[TableName] (**String** dbName, **String** conditions)
@GetMapping("/get[TableName]") String	get[TableName] (**String** dbName)

void setController (**Controller** controller)

- Sets *this.controller* to *controller.*

String print (**String** message)

- Prints a message.

String add[TableName] (**String** dbName, [*attributes*])

- [TableName] should be replaced with the actual table name (ex. addAnimal)
- [*attributes*] should be replaced with the actual table's attributes
- Ex.: Animal's [*attributes*] are **Integer** idAnimal, **String** name, **String** date, **Integer** animalType.

String remove[TableName] (**String** dbName, **String** conditions)

- [TableName] should be replaced with the actual table name (ex. removeAnimal)
- *conditions* can either be of format "attributeName=value" or just "value", but the user has to specify the exact order in which the attributes are defined in the class if the attribute name is not specified.

String get[TableName] (**String** dbName)

- [TableName] should be replaced with the actual table name (ex. getAnimals)