

Supplementary material for Enhancing Student Engagement and Learning in Database Programming Through Active Learning Strategies

--

Program-Building Cards Activity

This document provides additional information to facilitate replication of the program-building card (PBC) activity described in the paper.

1. Main Idea

The Program-Building Cards (PBC) activity is a gamified exercise designed to help students grasp the algorithmic structure of JDBC programs without being overwhelmed by syntax. Students, working in small groups, are given modular pseudocode-style cards that represent the essential steps of database interaction with JDBC. By arranging these cards into meaningful sequences, students can practice constructing database programs in a highly visual and interactive way.

2. Goal

The primary goal of the activity is for students to abstract away from the concrete syntax of the JDBC library and instead focus on the underlying process of database interaction. By doing so, students reinforce their procedural understanding, learn to generalize their knowledge, reinforcing procedural understanding and boosting group collaboration and interaction.

3. Schedule

The activity is integrated into the core of the JDBC module and is performed twice:

- **First round** – focusing on database information retrieval.
- **Second round** – focusing on database modification tasks.

Both rounds take place during the third and fourth JDBC lecture sessions, when students have acquired a basic knowledge of the JDBC library. This ensures that they can meaningfully engage with the task without requiring additional introductory explanations.

4. Setting

The instructor prepares a set of PBCs in advance. Each card corresponds to a key instruction involved in accessing a database with JDBC. The cards are deliberately:

- **Pseudocode-based**, not exact JDBC syntax, to encourage conceptual focus.
- **Modular**, broken down into small components that can be recombined in different ways.
- **Color-coded by category**, to support quick recognition and facilitate group discussion.

This design helps students concentrate on the process of constructing a database program (the logical sequence of steps) rather than the detailed syntax, which can later be studied in more depth.

5. Development

On each day the activity is carried out, students are organized into **groups of three**, primarily based on seating proximity to avoid classroom disruptions. Each group receives one set of program-building cards and is **given 5–8 minutes to solve a problem** projected on the board by the instructor.

The development proceeds as follows:

- **Problem presentation** – The instructor presents a short task (e.g., retrieving data from a database or updating a record).
- **Group discussion and construction** – Students work collaboratively, assembling the PBCs into what they believe is the correct sequence. During this phase, the instructor circulates, observing interactions, answering questions, and encouraging equal participation.
- **Solution assembly** – Each group finalizes its proposed sequence.
- **Teacher-guided correction** – Once time is up, the instructor gradually projects the solution, encouraging students to compare it with their group's result. This step is interactive: students are invited to justify their choices, reflect on alternative solutions, and correct misunderstandings.

This process not only reinforces procedural knowledge but also fosters peer dialogue and collaboration.

Note for readers: To facilitate reproducibility, **Appendix 1** provides a first version of the program-building cards, while **Appendix 2** includes examples of use. These materials are intended to help instructors replicate the activity in their own classrooms and adapt it to different database contexts.

4. Suggestions for Instructors

- **Time limit:** Give students 5–8 minutes per problem to encourage focus.
- **Flexibility:** Cards can be rearranged or extended depending on the complexity of the task.
- **Customization:** Instructors may adapt the wording or add visual symbols.
- **Discussion:** The post-activity review is crucial. Encourage students to compare their solution with the projected one and reflect on errors or alternative approaches.

Appendix 1. First Version of Program-Building Cards (PBC)

This appendix provides a **first version of the program-building cards** (PBC) used in the activity. The cards are expressed in simplified pseudocode to emphasize procedural understanding while removing syntax-related difficulties.

The intention is not to cover every detail of JDBC programming, but to provide a reusable set of modular building blocks. Instructors may adapt or expand the cards according to their course needs.

1. Structure of the Card Set

The cards are organized into **categories**, covering both fundamental steps in JDBC programming and general programming constructs.

A. Variable declaration

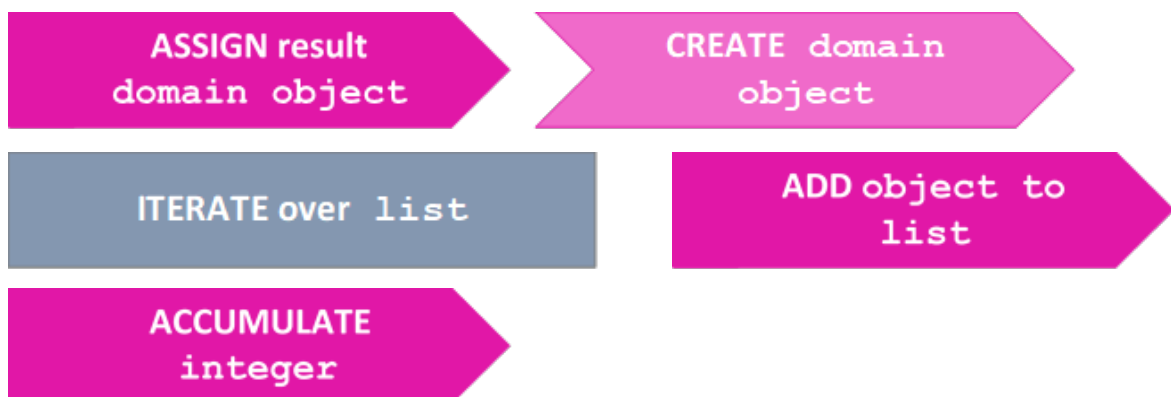
Card representing the creation of key objects.

DECLARE variables

B. General Programming Results

This category also includes programming elements that are not specific to JDBC but are commonly required when implementing methods. These blocks provide flexibility to model general control structures and data handling tasks, such as iterating over lists, assigning results, or accumulating values.

- **Create domain object:** Instantiates an object representing a row in the database.
- **Assign result domain object:** assigns an object representing a domain object.
- **Iterate over list:** loops through each element in a list
- **Add object to list:** stores the object in a collection if multiple results are expected.
- **Accumulate integer:** for statements that return an integer (e.g., updates, inserts, deletes), stores the cumulative effect or count.



C. Returning result

Card that symbolizes returning a value.

RETURN result

D. Transaction management

These blocks explicitly highlight how to ensure atomicity in database operations.

- **setAutoCommit to false**
- **commit**
- **rollback**

setAutoCommit to false

commit

rollback

E. Exception handling

These blocks reinforce good practices of robustness and error control:

- **try**: marks the start of a protected block.
- **catch**: standard error handling.
- **finally**: ensures resources (specially, connections) are closed regardless of success or failure.
- **throw ExceptionDeAplicacion**: example of raising a custom exception.

try

catch

finally

THROW ExceptionDeAplicacion

F. Initialization and Setup

In this first version, only the creation of the database connection has been considered, as it represents the essential entry point for any JDBC program. However, additional setup steps could also be included if desired, such as:

- Load driver – explicitly registering the JDBC driver.
- Define database URL – specifying the path and parameters of the target database.
- Set credentials (username, password) – providing the necessary authentication details.

These steps were left out for simplicity in the initial design but can easily be incorporated into future versions of the program-building cards to provide a more complete setup process.

CREATE Connection

G. SQL Statements (SELECT and Data Modification)

In this first version of the program-building cards, example blocks have been provided for SELECT statements and data modification statements.

- **SELECT statement:** allow students to practice retrieving data from tables.
- **Data modification statement** (INSERT, UPDATE, DELETE): enable students to add, modify, or remove records.

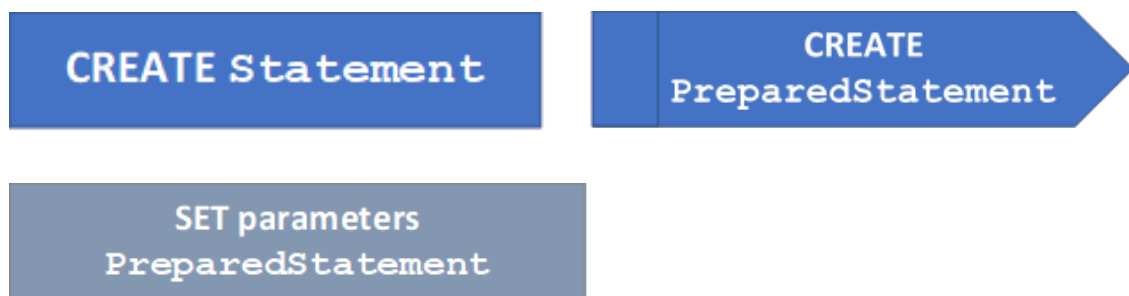
Future versions of the cards could include additional SQL statements or more complex query patterns to expand the scope of exercises.



H. Statement Preparation

These blocks set up the `PreparedStatement` or `Statement` with the SQL query (“SQL Statements”) and any required parameters.

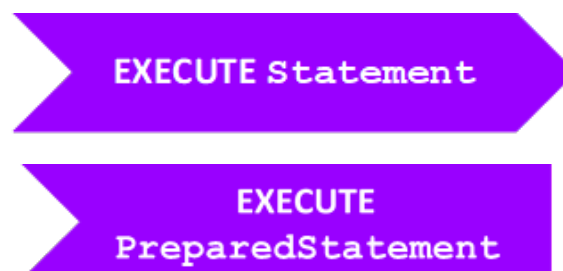
- **Create Statement**
- **Create PreparedStatement** (with SELECT or DATA MODIFICATION statements, showed previously)
- **Set parameters of the PreparedStatement**



I. Execution

These blocks allow running the `Statements` and `PreparedStatement`s to perform the database operation, returning a `ResultSet` for queries or an integer for updates/inserts/deletes. They should be used in combination with the corresponding “SQL Statements” and “ResultSet Creation and Processing Results”.

- **Execute Statement**
- **Execute PreparedStatement**

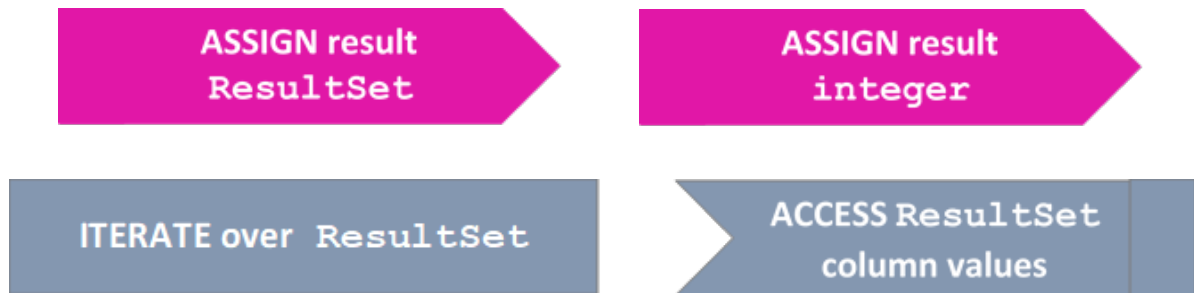


J. ResultSet Creation and Processing Results

This block covers the steps needed to handle query results after executing a database operation.

- **Assign result ResultSet:** assigns a `ResultSet` object after executing a select statement.
- **Assign result integer:** assigns an integer value after executing a modification statement.
- **Access ResultSet column values:** Retrieve the values of individual columns from the current row, either by column name or index.

Iterate over ResultSet: loop through each row (`while (rs.next())`).



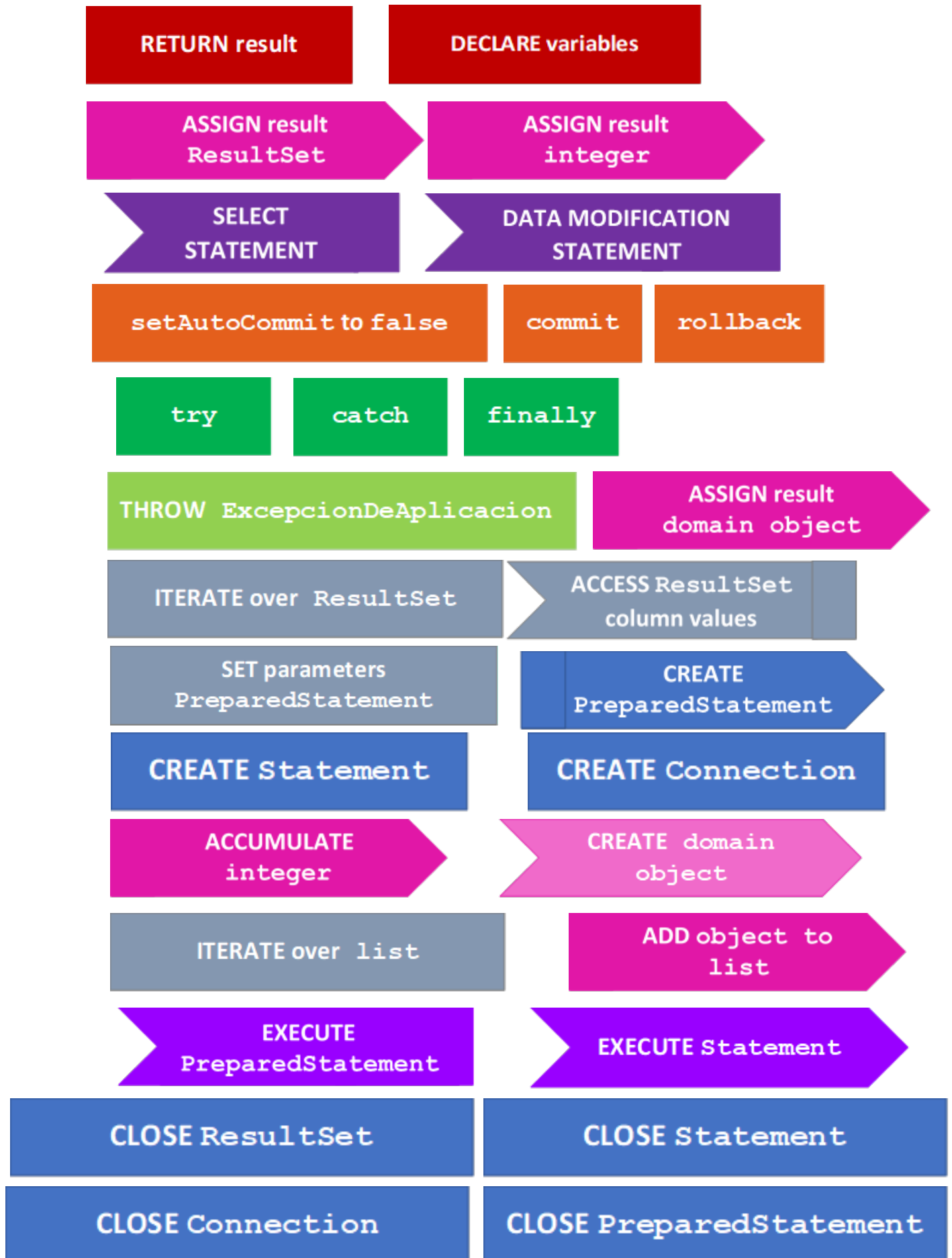
K. Closing Resources

This block ensures that database resources (`ResultSet`, `Statement`, `Connection`) are properly closed to avoid memory leaks and keep the application efficient.

- Close Result set
- Close Statement/Prepared Statement
- Close Connection



2. The Overall Card Set



Appendix 2. Examples of use

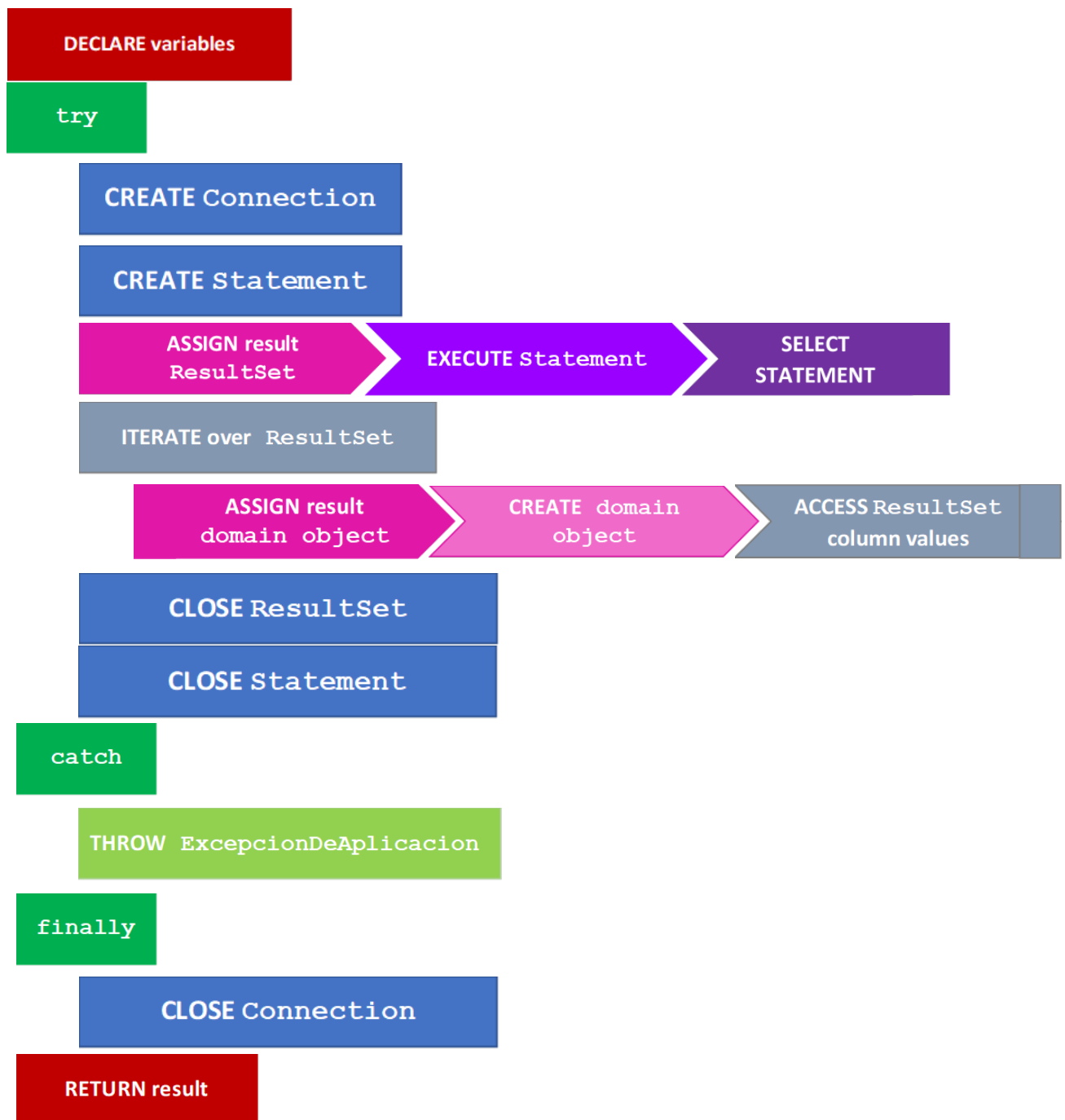
This appendix provides examples of how the program-building cards can be used for both query and modification tasks, illustrating typical classroom exercises and demonstrating how the cards support students in constructing correct JDBC programs.

1. Examples of Use: Retrieving Data

Problem projected by instructor: *Using the program-building cards, design the structure of a method for a database that stores product orders made by customers of a specific company. The method should do:*

- *Given the identifier of a product item,*
- *Return the corresponding **Item** object, created from the information stored in the database for that item within the context of customer orders.*
- *You should assume that the method declares it may throw an **ExcepcionDeAplicacion**.*
- *The method should be implemented using **Statements instead of PreparedStatement**s.*

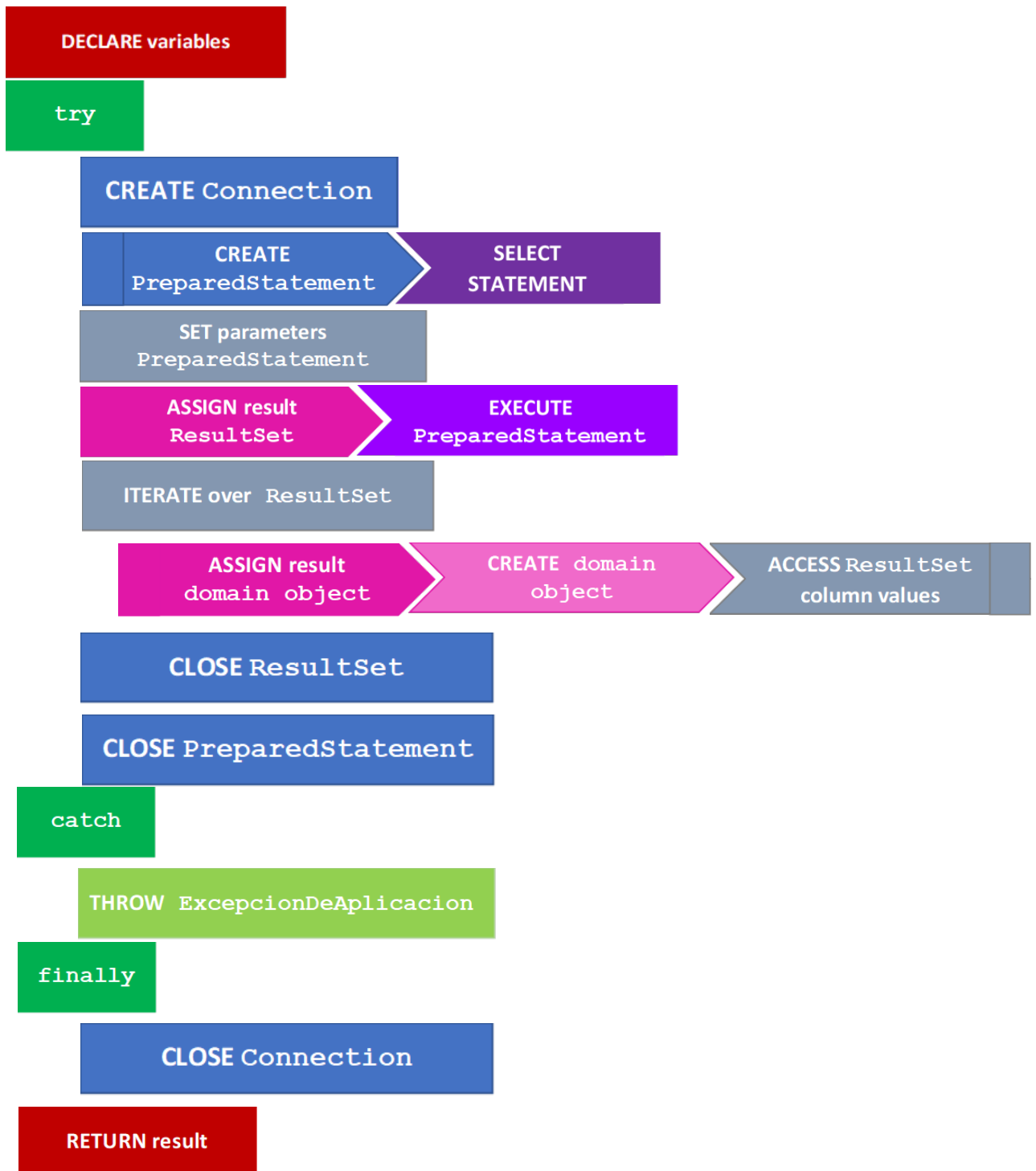
Correct sequence of cards (see below):



Problem projected by instructor: *Using the program-building cards, design the structure of a method for a database that stores product orders made by customers of a specific company. The method should do:*

- *Given the identifier of a product item,*
- *Return the corresponding **Item** object, created from the information stored in the database for that item within the context of customer orders.*
- *You should assume that the method declares it may throw an **ExcepcionDeAplicacion**.*
- *The method should be implemented using **PreparedStatement**.*

Correct sequence of cards:

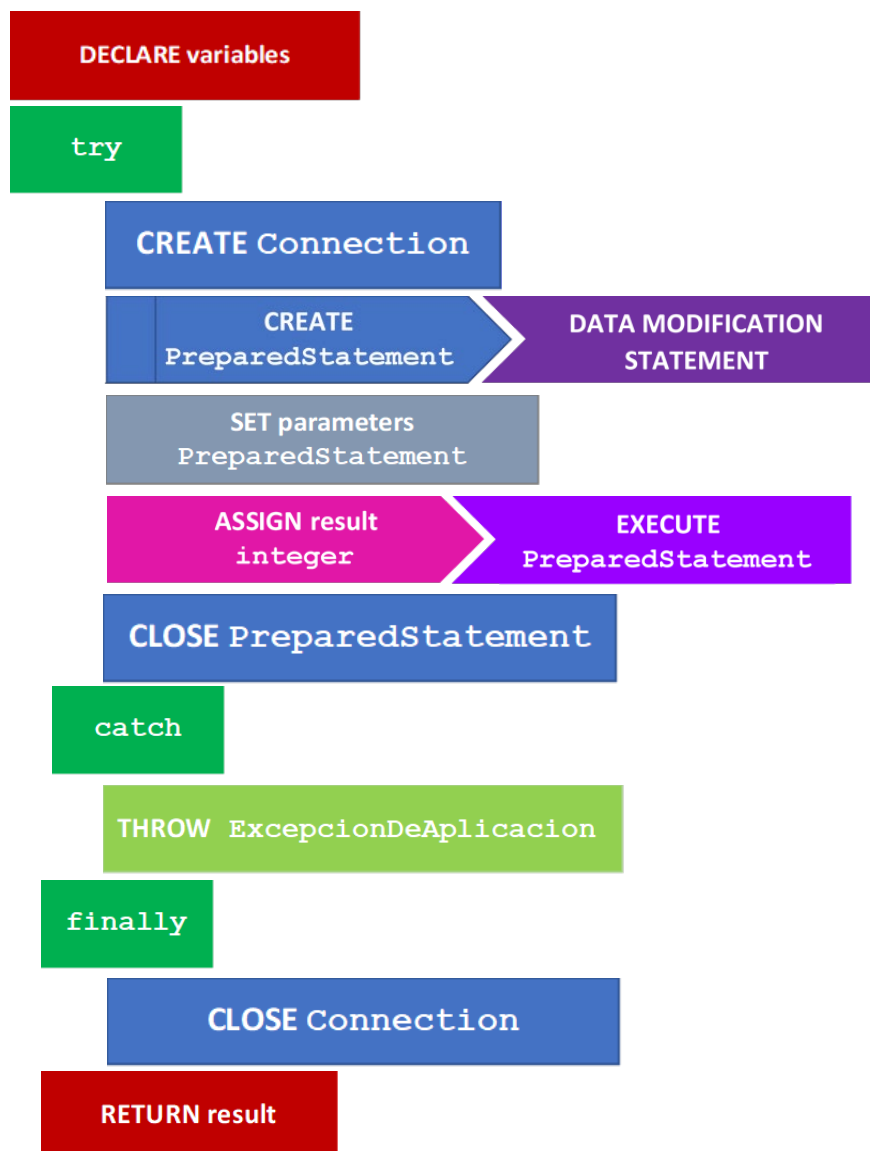


2. Examples of Use: Modifying Data

Problem projected by instructor: *Using the program-building cards, design the structure of a method for the previously described database, which also stores information about the company's employees. The method should do:*

- *Increase the salary of all employees in the Employee table by a specified percentage.*
- *If any error occurs during the transaction, the program will terminate without applying any salary changes (all or nothing).*
- *The program will return the number of employees whose salary was successfully updated.*
- *You should assume that the method declares it may throw an ExcepcionDeAplicacion.*
- *The method should be implemented using **PreparedStatements**.*

Correct sequence of cards (see below):



Problem projected by instructor: *Using the program-building cards, design the structure of a method for the previously described database, which also stores information about the company's employees. The method should do:*

- *Given a list of employee IDs...*
- *...deletes from the database, i.e., from the Employee table, the corresponding employees (only those employees).*
- *If any error occurs during the transaction, the program will terminate without deleting any employee (all or nothing).*
- *The program will return the number of employees deleted from the database.*
- *It may happen that some of the employee identifiers provided do not correspond to any employee in the database.*
- *You should assume that the method declares it may throw an **ExcepcionDeAplicacion**.*
- *The method should be implemented using **PreparedStatements**.*

Correct sequence of cards (see below):

