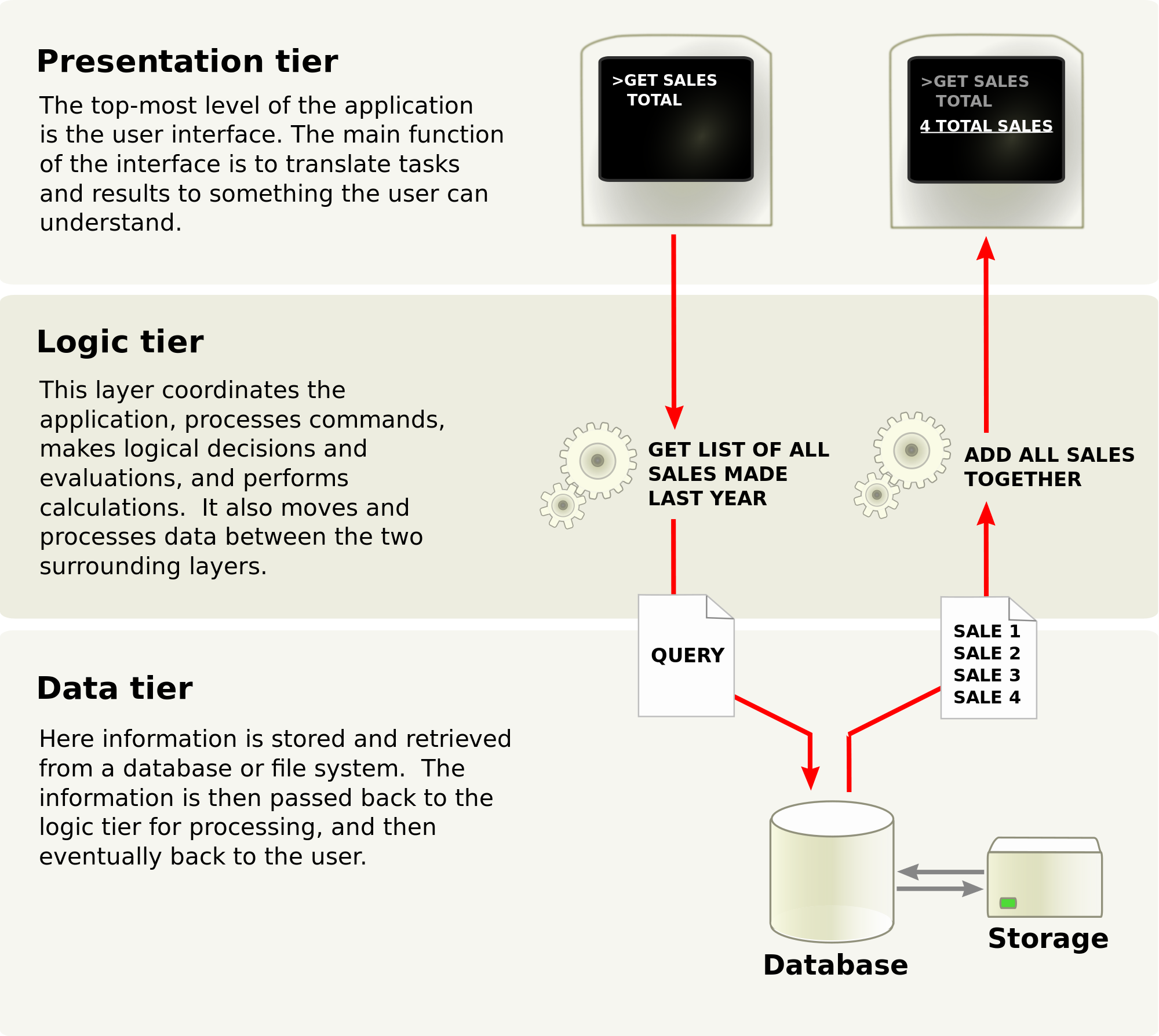
Java Logic Simulator [Desktop GUI Application]

* GUI Libraries
* SWT Graphics Library by Eclipse (IBM created it as an alternative for Swing)
  + Java2s Tutorials:  
    <http://www.java2s.com/Tutorial/Java/0280__SWT/Catalog0280__SWT.htm>
  + Eclipse Official Tutorials and tools:  
    <http://www.eclipse.org/swt/>
  + Book: The Definitive Guide to SWT and JFACE  
    <http://e-lib.kgsu.ru/books/the-definitive-guide-to-swt-and-jface.zip>
  + JFace UI toolkit based on SWT
  + Problem to make it cross-platform, i.e. packaging all native DLL files with your app in a JAR file.  
    Source can be found here:  
    <http://stackoverflow.com/questions/2037220/how-can-i-creating-executable-jar-with-swt-that-runs-on-all-platforms>
  + Eclipse WindowBuilder (GUI Builder) for SWT and Swing:  
    <https://eclipse.org/windowbuilder/>
* Swing and SwingX (reported as buggy and slow)
* Other Resources for GUI Libraries may be found Here:  
  <http://stackoverflow.com/questions/7358775/java-gui-frameworks-what-to-choose-swing-swt-awt-swingx-jgoodies-javafx>
* Diagram Building Tools
  + Draw.io (integrate with Google Docs)
  + Google Docs
  + UML Tools like StarUML
* Java Automated Testing Tools
  + Abbot: <http://abbot.sourceforge.net/doc/overview.shtml>
  + Other 8 useful tools: <https://blog.idrsolutions.com/2015/02/8-useful-java-testing-tools-frameworks-programmers-developers-coders/>
* Java Book Collection: <http://e-lib.kgsu.ru/swapj-books.html>
* Behavioral and Structural Diagrams using UML
  + Behavioral
    - Use-case Diagram
    - Sequence Diagram
    - Activity Diagram
  + Structural
    - Class Diagram
    - Object Diagram
    - Component Diagram
  + Learning Resources:
    - Tutorials Point: <http://www.tutorialspoint.com/uml/index.htm>
    - Source Making: <https://sourcemaking.com/uml>
* N-Tier Architecture



* + Illustration Video:  
    <https://www.youtube.com/watch?v=KlHvRKSH4pk>
  + Article about N-Tier  
    <http://www.codeproject.com/Articles/430014/N-Tier-Architecture-and-Tips>
* MVC Design Pattern (Model, View and Control)
  + Different between N-Tier and MVC:  
    <https://www.quora.com/What-is-the-difference-between-MVC-and-N-Tiered-architecture>
  + Also: N-tier architecture usually has each layer separated by the network. I.E. the presentation layer is on some web servers, then that talks to backend app servers over the network for business logic, then that talks to a database server, again over the network, and maybe the app server also calls out to some remote services (say Authorize.net for payment processing).

MVC is a programming design pattern where different portions of code are responsible for representing the Model, View, and controller in some application. These two things are related because, for instance the Model layer may have an internal implementation that calls a database for storing and retrieving data. The controller may reside on the webserver, and remotely call appservers to retrieve data. MVC abstracts away the details of how the architecture of an app is implemented.

N-tier just refers to the physical structure of an implementation. These two are sometimes confused because an MVC design is often implemented using an N-tier architecture.

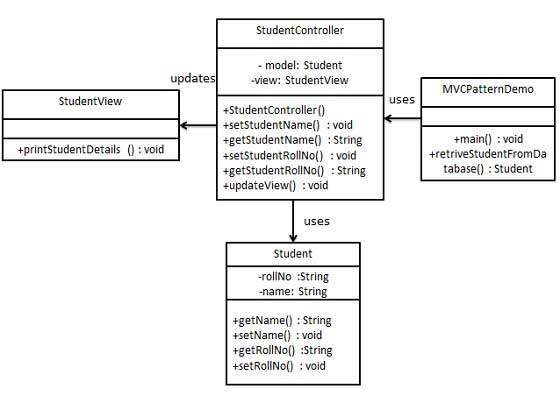
* + Useful Article Demonstrate How to Implement MVC with Java:  
    <http://www.codeproject.com/Articles/879896/Programming-in-Java-using-the-MVC-architecture>
  + Example from Tutorials Point: (It's Copied in the following pages)  
    <http://www.tutorialspoint.com/design_pattern/mvc_pattern.htm>

MVC Pattern stands for Model-View-Controller Pattern. This pattern is used to separate application's concerns.

* **Model** - Model represents an object or JAVA POJO carrying data. It can also have logic to update controller if its data changes.
* **View** - View represents the visualization of the data that model contains.
* **Controller** - Controller acts on both model and view. It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.

Implementation

We are going to create a *Student* object acting as a model.*StudentView* will be a view class which can print student details on console and *StudentController* is the controller class responsible to store data in *Student* object and update view*StudentView* accordingly.

*MVCPatternDemo*, our demo class, will use *StudentController* to demonstrate use of MVC pattern.

Step 1

Create Model.

*Student.java*

public class Student {

private String rollNo;

private String name;

public String getRollNo() {

return rollNo;

}

public void setRollNo(String rollNo) {

this.rollNo = rollNo;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

}

Step 2

Create View.

*StudentView.java*

public class StudentView {

public void printStudentDetails(String studentName, String studentRollNo){

System.out.println("Student: ");

System.out.println("Name: " + studentName);

System.out.println("Roll No: " + studentRollNo);

}

}

Step 3

Create Controller.

*StudentController.java*

public class StudentController {

private Student model;

private StudentView view;

public StudentController(Student model, StudentView view){

this.model = model;

this.view = view;

}

public void setStudentName(String name){

model.setName(name);

}

public String getStudentName(){

return model.getName();

}

public void setStudentRollNo(String rollNo){

model.setRollNo(rollNo);

}

public String getStudentRollNo(){

return model.getRollNo();

}

public void updateView(){

view.printStudentDetails(model.getName(), model.getRollNo());

}

}

Step 4

Use the *StudentController* methods to demonstrate MVC design pattern usage.

*MVCPatternDemo.java*

public class MVCPatternDemo {

public static void main(String[] args) {

//fetch student record based on his roll no from the database

Student model = retriveStudentFromDatabase();

//Create a view : to write student details on console

StudentView view = new StudentView();

StudentController controller = new StudentController(model, view);

controller.updateView();

//update model data

controller.setStudentName("John");

controller.updateView();

}

private static Student retriveStudentFromDatabase(){

Student student = new Student();

student.setName("Robert");

student.setRollNo("10");

return student;

}

}

Step 5

Verify the output.

Student:

Name: Robert

Roll No: 10

Student:

Name: John

Roll No: 10

* Agile Development Process
  + Using Github Issues
    - Read this:  
      <http://www.position-absolute.com/articles/agile-workflow-with-github-issues/>
    - Tools:  
      <http://liftux.com/posts/using-github-issues-project-management/>
    - Waffle Tool [Free for Public Github repos]  
      <https://waffle.io/>
  + Backlog
    - A backlog is a list of features or technical tasks which the team maintains and which, at a given moment, are known to be necessary and sufficient to complete a project or a release: - **See more** at: <http://guide.agilealliance.org/guide/backlog.html>
    - More Resources:  
      <https://www.atlassian.com/agile/backlogs/>
  + JIRA Software Development Tool [**More Advanced for Big Agile Projects**]
    - What is JIRA: <https://www.youtube.com/watch?time_continue=6&v=8KPoZ5g8NqU>
    - Website: <https://www.atlassian.com/software/jira>
    - Tutorials Can be found online on torrent sites.
* Mixed Waterfall and Agile Development Process

