C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\app\App.java

App

```java
package taskmanagement.app;

import taskmanagement.ui.MainFrame;
import taskmanagement.ui.styles.AppTheme;

import javax.swing.SwingUtilities;

/**
 * Application entry point for launching the Tasks Management Application UI.
 * <p>
 * This class ensures the Swing user interface is started on the Event Dispatch Thread (EDT),
 * applies global theme defaults, and displays the main application window.
 * </p>
 */
public final class App {

    /**
     * Private constructor to prevent instantiation.
     * This is a utility class and should not be instantiated.
     */
    private App() { }

    /**
     * Main program entry point.
     * <p>
     * Responsibilities:
     * <ul>
     *   <li>Schedules UI initialization on the Event Dispatch Thread (EDT).</li>
     *   <li>Applies global Swing theme defaults.</li>
     *   <li>Sets a default uncaught exception handler.</li>
     *   <li>Creates and shows the {@link MainFrame}.</li>
     * </ul>
     * </p>
     *
     * @param args command-line arguments (not used)
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            AppTheme.applyAccentDefaults();
            Thread.setDefaultUncaughtExceptionHandler((t, e) -> e.printStackTrace(System.err));
            MainFrame frame = new MainFrame();
            frame.setVisible(true);
        });
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\AddTaskCommand.java

AddTaskCommand

```java
package taskmanagement.application.viewmodel.commands;

import taskmanagement.domain.ITask;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.util.Objects;

/**
 * A command that adds a new task to the system.
 * <p>
 * This command follows the Command design pattern:
 * <ul>
 *   <li>{@link #execute()} adds the task using the DAO and captures the generated ID.</li>
 *   <li>{@link #undo()} deletes the task created by {@code execute()} using the captured ID.</li>
 * </ul>
 * </p>
 */
public final class AddTaskCommand implements Command {

    private final ITasksDAO dao;
    private final ITask taskToAdd;
    private Integer createdId;

    /**
     * Constructs a new {@code AddTaskCommand}.
     *
     * @param dao       the tasks DAO used to persist the task (must not be {@code null})
     * @param taskToAdd the task instance to be added (must not be {@code null})
     * @throws NullPointerException if {@code dao} or {@code taskToAdd} is {@code null}
     */
    public AddTaskCommand(ITasksDAO dao, ITask taskToAdd) {
        this.dao = Objects.requireNonNull(dao, "dao");
        this.taskToAdd = Objects.requireNonNull(taskToAdd, "taskToAdd");
    }

    /**
     * Returns the name of this command.
     *
     * @return a string describing the command
     */
    @Override
    public String name() {
        return "Add Task";
    }

    /**
     * Executes the command by adding the task via the DAO.
     * <p>
     * After execution, the generated ID must be reflected in {@link ITask#getId()}.
     * If the DAO does not assign a valid ID, a {@link TasksDAOException} is thrown.
     * </p>
     *
     * @throws TasksDAOException if the DAO fails to add the task or returns an invalid ID
     */
    @Override
    public void execute() throws TasksDAOException {
        dao.addTask(taskToAdd);
        createdId = taskToAdd.getId();
```

AddTaskCommand

```java
        if (createdId == null || createdId < 0) {
            throw new TasksDAOException(
                    "DAO did not assign a valid id to the added task (got: " + createdId + ")"
            );
        }
    }

    /**
     * Undoes the addition of the task by deleting it using the captured ID.
     *
     * @throws TasksDAOException if the captured ID is invalid or the DAO fails to delete the task
     */
    @Override
    public void undo() throws TasksDAOException {
        if (createdId == null || createdId <= 0) {
            throw new TasksDAOException("Cannot undo add: missing valid createdId");
        }
        dao.deleteTask(createdId);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\Command.java

Command

```java
package taskmanagement.application.viewmodel.commands;

import taskmanagement.domain.exceptions.ValidationException;
import taskmanagement.persistence.TasksDAOException;

/**
 * Represents a reversible application action following the Command pattern.
 * <p>
 * Each command encapsulates all data required to perform an operation and
 * to safely undo it.
 * </p>
 */
public interface Command {

    /**
     * Returns a short human-readable name of this command.
     * Useful for menus, logs, or debugging.
     *
     * @return the command name
     */
    String name();

    /**
     * Executes the command.
     * <p>
     * Implementations must be idempotent with respect to redo,
     * meaning calling {@code execute()} again after an undo should
     * have the same effect as the initial execution.
     * </p>
     *
     * @throws TasksDAOException     if a persistence-related error occurs
     * @throws ValidationException   if domain validation fails
     */
    void execute() throws TasksDAOException, ValidationException;

    /**
     * Undoes the last successful execution of this command.
     *
     * @throws TasksDAOException     if a persistence-related error occurs
     * @throws ValidationException   if domain validation fails
     */
    void undo() throws TasksDAOException, ValidationException;
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\CommandException.java

CommandException

```java
package taskmanagement.application.viewmodel.commands;

/**
 * Exception type representing errors that occur during command execution or undo.
 * <p>
 * This exception wraps lower-level exceptions (such as DAO or validation errors)
 * so that the ViewModel can propagate a single checked exception type to the UI.
 * </p>
 */
public class CommandException extends Exception {

    /**
     * Constructs a new {@code CommandException} with the specified detail message.
     *
     * @param message the detail message
     */
    public CommandException(String message) {
        super(message);
    }

    /**
     * Constructs a new {@code CommandException} with the specified detail message
     * and cause.
     *
     * @param message the detail message
     * @param cause   the underlying cause of this exception
     */
    public CommandException(String message, Throwable cause) {
        super(message, cause);
    }

    /**
     * Constructs a new {@code CommandException} with the specified cause.
     *
     * @param cause the underlying cause of this exception
     */
    public CommandException(Throwable cause) {
        super(cause);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\CommandStack.java

CommandStack

```java
package taskmanagement.application.viewmodel.commands;

import taskmanagement.application.viewmodel.events.Property;
import taskmanagement.domain.exceptions.ValidationException;
import taskmanagement.persistence.TasksDAOException;

import java.util.ArrayDeque;
import java.util.Deque;
import java.util.Objects;

/**
 * Maintains undo and redo stacks for executed commands.
 * <p>
 * This class implements the Command design pattern infrastructure,
 * allowing commands to be executed, undone, and redone.
 * It provides both:
 * <ul>
 *   <li>Pull-style status checks via {@link #canUndo()} and {@link #canRedo()}.</li>
 *   <li>Push-style observables for UI binding via {@link #canUndoProperty()} and {@link #canRedoProperty()}.</li>
 * </ul>
 * </p>
 */
public final class CommandStack {

    private final Deque<Command> undoStack = new ArrayDeque<>();
    private final Deque<Command> redoStack = new ArrayDeque<>();

    private final Property<Boolean> canUndoProp = new Property<>(false);
    private final Property<Boolean> canRedoProp = new Property<>(false);

    /**
     * Executes the given command, pushes it onto the undo stack,
     * and clears the redo stack.
     *
     * @param command the command to execute (must not be {@code null})
     * @throws CommandException if execution fails
     */
    public void execute(Command command) throws CommandException {
        Objects.requireNonNull(command, "command");
        try {
            command.execute();
            undoStack.push(command);
            redoStack.clear();
            refreshFlags();
        } catch (TasksDAOException | ValidationException ex) {
            throw new CommandException("Failed to execute command: " + command.name(), ex);
        }
    }

    /**
     * Undoes the last executed command and moves it to the redo stack.
     * Does nothing if there is no command to undo.
     *
     * @throws CommandException if undo fails
     */
    public void undo() throws CommandException {
        if (undoStack.isEmpty()) {
            return;
        }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\CommandStack.java

CommandStack

```java
        Command c = undoStack.pop();
        try {
            c.undo();
            redoStack.push(c);
            refreshFlags();
        } catch (TasksDAOException | ValidationException ex) {
            undoStack.push(c);
            refreshFlags();
            throw new CommandException("Failed to undo command: " + c.name(), ex);
        }
    }

    /**
     * Redoes the last undone command and moves it back to the undo stack.
     * Does nothing if there is no command to redo.
     *
     * @throws CommandException if redo fails
     */
    public void redo() throws CommandException {
        if (redoStack.isEmpty()) {
            return;
        }
        Command c = redoStack.pop();
        try {
            c.execute();
            undoStack.push(c);
            refreshFlags();
        } catch (TasksDAOException | ValidationException ex) {
            redoStack.push(c);
            refreshFlags();
            throw new CommandException("Failed to redo command: " + c.name(), ex);
        }
    }

    /**
     * Indicates whether an undo operation is available.
     *
     * @return {@code true} if there is at least one command to undo
     */
    public boolean canUndo() {
        return !undoStack.isEmpty();
    }

    /**
     * Indicates whether a redo operation is available.
     *
     * @return {@code true} if there is at least one command to redo
     */
    public boolean canRedo() {
        return !redoStack.isEmpty();
    }

    /**
     * Provides an observable property for the undo availability flag.
     * Useful for enabling or disabling undo-related UI controls.
     *
     * @return the observable undo property
     */
    public Property<Boolean> canUndoProperty() {
        return canUndoProp;
    }

    /**
     * Provides an observable property for the redo availability flag.
     * Useful for enabling or disabling redo-related UI controls.
     *
     * @return the observable redo property
     */
    public Property<Boolean> canRedoProperty() {
        return canRedoProp;
    }

    /**
     * Clears both the undo and redo stacks and refreshes availability flags.
     * Typically used when resetting the application state.
     */
    public void clear() {
        undoStack.clear();
        redoStack.clear();
        refreshFlags();
    }

    /**
     * Returns the number of commands currently available for undo.
     *
     * @return the size of the undo stack
     */
    public int undoCount() {
        return undoStack.size();
    }

    /**
     * Returns the number of commands currently available for redo.
     *
     * @return the size of the redo stack
     */
    public int redoCount() {
        return redoStack.size();
    }

    /**
     * Updates the observable flags for undo and redo availability.
     */
    private void refreshFlags() {
        canUndoProp.setValue(!undoStack.isEmpty());
        canRedoProp.setValue(!redoStack.isEmpty());
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\DeleteTaskCommand.java

DeleteTaskCommand

```java
package taskmanagement.application.viewmodel.commands;

import taskmanagement.domain.ITask;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.util.Objects;

/**
 * Command that deletes a task from the DAO and supports undo by restoring it.
 * <p>
 * Implements the Command design pattern:
 * <ul>
 *   <li>{@link #execute()} removes the task from persistence.</li>
 *   <li>{@link #undo()} restores the previously deleted task snapshot.</li>
 * </ul>
 * </p>
 */
public final class DeleteTaskCommand implements Command {

    private final ITasksDAO dao;
    private final ITask deletedSnapshot;

    /**
     * Constructs a new {@code DeleteTaskCommand}.
     *
     * @param dao             the tasks DAO used for persistence (must not be {@code null})
     * @param deletedSnapshot snapshot of the task to be deleted (must not be {@code null})
     * @throws NullPointerException if {@code dao} or {@code deletedSnapshot} is {@code null}
     */
    public DeleteTaskCommand(ITasksDAO dao, ITask deletedSnapshot) {
        this.dao = Objects.requireNonNull(dao, "dao");
        this.deletedSnapshot = Objects.requireNonNull(deletedSnapshot, "deletedSnapshot");
    }

    /**
     * Returns the human-readable name of this command.
     *
     * @return the command name
     */
    @Override
    public String name() {
        return "Delete Task";
    }

    /**
     * Executes the deletion of the task identified by its ID.
     *
     * @throws TasksDAOException if the DAO fails to delete the task
     */
    @Override
    public void execute() throws TasksDAOException {
        dao.deleteTask(deletedSnapshot.getId());
    }

    /**
     * Restores the previously deleted task by re-inserting its snapshot.
     *
     * @throws TasksDAOException if the DAO fails to re-add the task
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\DeleteTaskCommand.java

DeleteTaskCommand

```java
     */
    @Override
    public void undo() throws TasksDAOException {
        dao.addTask(deletedSnapshot);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\MarkStateCommand.java

MarkStateCommand

```java
package taskmanagement.application.viewmodel.commands;

import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.exceptions.ValidationException;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.util.Objects;

/**
 * Command that changes the lifecycle state of a task.
 * <p>
 * Implements the Command design pattern:
 * <ul>
 *   <li>{@link #execute()} updates the task to the target state using the DAO.</li>
 *   <li>{@link #undo()} restores the original state using the stored snapshot.</li>
 * </ul>
 * </p>
 * <p>
 * Validation of whether the transition is legal must be performed
 * externally by the caller (e.g., in the ViewModel).
 * </p>
 */
public final class MarkStateCommand implements Command {

    /**
     * Factory for producing a new immutable task snapshot with a modified state.
     */
    @FunctionalInterface
    public interface TaskFactory {
        /**
         * Creates a copy of the given task with a new state applied.
         *
         * @param src       the original task
         * @param newState  the new state to assign
         * @return a new task snapshot with the updated state
         * @throws ValidationException if the new state is invalid for the task
         */
        ITask copyWithState(ITask src, TaskState newState) throws ValidationException;
    }

    private final ITasksDAO dao;
    private final ITask beforeSnapshot;
    private final TaskState targetState;
    private final TaskFactory factory;

    private ITask afterSnapshot;

    /**
     * Constructs a new {@code MarkStateCommand}.
     *
     * @param dao            the DAO for persistence operations (must not be {@code null})
     * @param beforeSnapshot snapshot of the task before modification (must not be {@code null})
     * @param targetState    the target state to apply (must not be {@code null})
     * @param factory        factory for creating modified task snapshots (must not be {@code null})
     * @throws NullPointerException if any argument is {@code null}
     */
    public MarkStateCommand(ITasksDAO dao, ITask beforeSnapshot, TaskState targetState, TaskFactory factory) {
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\MarkStateCommand.java

MarkStateCommand

```java
        this.dao = Objects.requireNonNull(dao, "dao");
        this.beforeSnapshot = Objects.requireNonNull(beforeSnapshot, "beforeSnapshot");
        this.targetState = Objects.requireNonNull(targetState, "targetState");
        this.factory = Objects.requireNonNull(factory, "factory");
    }

    /**
     * Returns the human-readable name of this command.
     *
     * @return the command name including the target state
     */
    @Override
    public String name() {
        return "Mark State: " + targetState;
    }

    /**
     * Executes the command by updating the task to the target state.
     * If not already created, generates the "after" snapshot using the factory.
     *
     * @throws TasksDAOException     if the DAO update fails
     * @throws ValidationException   if the factory produces an invalid task
     */
    @Override
    public void execute() throws TasksDAOException, ValidationException {
        if (afterSnapshot == null) {
            afterSnapshot = factory.copyWithState(beforeSnapshot, targetState);
        }
        dao.updateTask(afterSnapshot);
    }

    /**
     * Undoes the state change by restoring the original snapshot.
     *
     * @throws TasksDAOException if the DAO update fails
     */
    @Override
    public void undo() throws TasksDAOException {
        dao.updateTask(beforeSnapshot);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\commands\UpdateTaskCommand.java

UpdateTaskCommand

```java
package taskmanagement.application.viewmodel.commands;

import taskmanagement.domain.ITask;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.util.Objects;

/**
 * Command that updates an existing task to a new snapshot.
 * <p>
 * Implements the Command design pattern:
 * <ul>
 *   <li>{@link #execute()} updates the task to the {@code afterSnapshot} state.</li>
 *   <li>{@link #undo()} restores the task to the {@code beforeSnapshot} state.</li>
 * </ul>
 * </p>
 */
public final class UpdateTaskCommand implements Command {

    private final ITasksDAO dao;
    private final ITask beforeSnapshot;
    private final ITask afterSnapshot;

    /**
     * Constructs a new {@code UpdateTaskCommand}.
     *
     * @param dao            the tasks DAO (must not be {@code null})
     * @param beforeSnapshot snapshot of the entity before update (must not be {@code null})
     * @param afterSnapshot  snapshot of the entity after update (must not be {@code null})
     * @throws NullPointerException if any argument is {@code null}
     */
    public UpdateTaskCommand(ITasksDAO dao, ITask beforeSnapshot, ITask afterSnapshot) {
        this.dao = Objects.requireNonNull(dao, "dao");
        this.beforeSnapshot = Objects.requireNonNull(beforeSnapshot, "beforeSnapshot");
        this.afterSnapshot = Objects.requireNonNull(afterSnapshot, "afterSnapshot");
    }

    /**
     * Returns the human-readable name of this command.
     *
     * @return the command name
     */
    @Override
    public String name() {
        return "Update Task";
    }

    /**
     * Executes the update by applying the {@code afterSnapshot}.
     * <p>
     * Ensures both snapshots belong to the same task ID before performing the update.
     * </p>
     *
     * @throws TasksDAOException if the IDs mismatch or the DAO update fails
     */
    @Override
    public void execute() throws TasksDAOException {
        if (beforeSnapshot.getId() != afterSnapshot.getId()) {
```

UpdateTaskCommand

```java
            throw new TasksDAOException(
                    "Mismatched ids between before/after snapshots: "
                            + beforeSnapshot.getId() + " vs " + afterSnapshot.getId()
            );
        }
        dao.updateTask(afterSnapshot);
    }

    /**
     * Undoes the update by restoring the {@code beforeSnapshot}.
     *
     * @throws TasksDAOException if the DAO update fails
     */
    @Override
    public void undo() throws TasksDAOException {
        dao.updateTask(beforeSnapshot);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\events\ObservableList.java

ObservableList

```java
package taskmanagement.application.viewmodel.events;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Objects;

/**
 * Observable list wrapper for UI binding.
 * <p>
 * This class does not implement the full {@link java.util.List} interface.
 * Instead, it manages an internal list and exposes methods for replacing,
 * clearing, and observing changes to the list as a whole.
 * Useful for scenarios such as refreshing a UI table when the list changes.
 * </p>
 *
 * @param <T> the element type of the list
 */
public final class ObservableList<T> {

    /**
     * Listener notified when the list reference or contents are changed.
     *
     * @param <T> element type
     */
    @FunctionalInterface
    public interface Listener<T> {
        /**
         * Invoked when the list has been updated.
         *
         * @param newSnapshot an immutable snapshot of the updated list
         */
        void onListChanged(List<T> newSnapshot);
    }

    private List<T> data = List.of();
    private final List<Listener<T>> listeners = new ArrayList<>();

    /**
     * Returns an immutable snapshot of the current list contents.
     *
     * @return the current immutable list
     */
    public List<T> get() {
        return data;
    }

    /**
     * Replaces the entire list with the provided items and notifies listeners.
     *
     * @param items the new list contents; if {@code null}, the list becomes empty
     */
    public void set(List<T> items) {
        List<T> newSnap = (items == null) ? List.of() : List.copyOf(items);
        if (!Objects.equals(this.data, newSnap)) {
            this.data = newSnap;
            fireChanged();
        }
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\events\ObservableList.java

ObservableList

```java
    /**
     * Clears the list contents and notifies listeners if it was not already empty.
     */
    public void clear() {
        if (!data.isEmpty()) {
            this.data = List.of();
            fireChanged();
        }
    }

    /**
     * Adds a listener to be notified when the list changes.
     * Duplicate additions are ignored.
     *
     * @param l the listener to add (must not be {@code null})
     */
    public void addListener(Listener<T> l) {
        Objects.requireNonNull(l, "listener must not be null");
        if (!listeners.contains(l)) {
            listeners.add(l);
        }
    }

    /**
     * Removes a listener if present.
     *
     * @param l the listener to remove
     */
    public void removeListener(Listener<T> l) {
        listeners.remove(l);
    }

    /**
     * Returns an immutable snapshot of the current listeners.
     *
     * @return an unmodifiable list of listeners
     */
    public List<Listener<T>> getListeners() {
        return Collections.unmodifiableList(listeners);
    }

    /**
     * Notifies all registered listeners of the current list contents.
     * <p>
     * Each listener receives the same immutable snapshot of the data.
     * Runtime exceptions from one listener do not prevent notification of others.
     * </p>
     */
    private void fireChanged() {
        List<Listener<T>> copy = List.copyOf(listeners);
        List<T> snap = data; // immutable snapshot
        for (Listener<T> l : copy) {
            try {
                l.onListChanged(snap);
            } catch (RuntimeException ex) {
                // exception suppressed to allow remaining listeners to be notified
            }
        }
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\events\Property.java

Property

```java
package taskmanagement.application.viewmodel.events;

import java.util.List;
import java.util.Objects;
import java.util.concurrent.CopyOnWriteArrayList;

/**
 * Observable property implementing the Observer pattern.
 * <p>
 * Listeners can subscribe via {@link #addListener(Listener)} and will be notified
 * whenever the property's value is set or explicitly refreshed.
 * </p>
 * <p>
 * Important: {@link #setValue(Object)} always notifies listeners,
 * even if the old and new values are equal according to {@link Objects#equals(Object, Object)}.
 * This ensures in-place updates (e.g., mutable objects) still trigger a refresh.
 * </p>
 *
 * @param <T> the type of the property's value (nullable)
 */
public final class Property<T> {

    /**
     * Listener interface for observing property changes.
     *
     * @param <T> the type of the observed value
     */
    @FunctionalInterface
    public interface Listener<T> {
        /**
         * Called when the property's value changes.
         *
         * @param oldValue the previous value (nullable)
         * @param newValue the new value (nullable)
         */
        void onChanged(T oldValue, T newValue);
    }

    private volatile T value;
    private final List<Listener<T>> listeners = new CopyOnWriteArrayList<>();

    /**
     * Constructs a new property with the given initial value.
     *
     * @param initial the initial value (nullable)
     */
    public Property(T initial) {
        this.value = initial;
    }

    /**
     * Returns the current value of this property.
     *
     * @return the current value (nullable)
     */
    public T getValue() {
        return value;
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\events\Property.java

Property

```java
    /**
     * Sets a new value and always notifies listeners, regardless of equality.
     *
     * @param newValue the new value (nullable)
     */
    public void setValue(T newValue) {
        T old = this.value;
        this.value = newValue;
        fireChanged(old, newValue);
    }

    /**
     * Sets a new value and notifies listeners only if the value has changed
     * according to {@link Objects#equals(Object, Object)}.
     *
     * @param newValue the new value (nullable)
     */
    public void setValueIfChanged(T newValue) {
        T old = this.value;
        if (!Objects.equals(old, newValue)) {
            this.value = newValue;
            fireChanged(old, newValue);
        }
    }

    /**
     * Notifies listeners of a change without modifying the current value.
     * Useful when the current value is mutable and has been updated in place.
     */
    public void fireChange() {
        fireChanged(this.value, this.value);
    }

    /**
     * Registers a new listener to be notified of property changes.
     * Duplicate additions are ignored.
     *
     * @param l the listener to add (must not be {@code null})
     */
    public void addListener(Listener<T> l) {
        listeners.add(Objects.requireNonNull(l, "listener must not be null"));
    }

    /**
     * Removes a previously registered listener.
     * Does nothing if the listener is not registered.
     *
     * @param l the listener to remove
     */
    public void removeListener(Listener<T> l) {
        listeners.remove(l);
    }

    /**
     * Notifies all registered listeners of the property change.
     *
     * @param oldValue the old value (nullable)
     * @param newValue the new value (nullable)
     */
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\events\Property.java

Property

```
    private void fireChanged(T oldValue, T newValue) {
        for (Listener<T> l : listeners) {
            try {
                l.onChanged(oldValue, newValue);
            } catch (RuntimeException ignored) {
                // Listener exceptions are suppressed to avoid breaking the chain.
            }
        }
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\ExportFormat.java

ExportFormat

```java
package taskmanagement.application.viewmodel;

/**
 * Enumeration of file export formats supported by the application.
 */
public enum ExportFormat {

    /**
     * Comma-separated values format.
     */
    CSV,

    /**
     * Plain text format.
     */
    TXT
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\sort\SortById.java

SortById

```java
package taskmanagement.application.viewmodel.sort;

import taskmanagement.domain.ITask;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Objects;

/**
 * Sorting strategy that orders tasks by their unique identifier.
 * <p>
 * This strategy serves as the default (identity-based) ordering
 * for {@link ITask} collections.
 * </p>
 */
public final class SortById implements SortStrategy {

    /**
     * Returns the human-readable display name of this sorting strategy.
     *
     * @return the display name for this strategy
     */
    @Override
    public String displayName() {
        return "ID (Default)";
    }

    /**
     * Returns a new list of tasks sorted by their {@code id} in ascending order.
     * The input list is not modified.
     *
     * @param items the list of tasks to be sorted (must not be {@code null})
     * @return a new list containing the tasks sorted by {@code id}
     * @throws NullPointerException if {@code items} is {@code null}
     */
    @Override
    public List<ITask> sort(List<ITask> items) {
        Objects.requireNonNull(items, "items");
        List<ITask> copy = new ArrayList<>(items);
        copy.sort(Comparator.comparingInt(ITask::getId));
        return copy;
    }

    /**
     * Returns the string representation of this sorting strategy,
     * which is the same as its display name.
     *
     * @return the display name string
     */
    @Override
    public String toString() {
        return displayName();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\sort\SortByState.java

SortByState

```java
package taskmanagement.application.viewmodel.sort;

import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Objects;

/**
 * Sorting strategy that orders tasks by their lifecycle state in the sequence:
 * <ul>
 *   <li>{@link TaskState#ToDo}</li>
 *   <li>{@link TaskState#InProgress}</li>
 *   <li>{@link TaskState#Completed}</li>
 * </ul>
 * <p>
 * Within the same state, tasks are further ordered by case-insensitive title,
 * and finally by task ID to ensure stable ordering.
 * </p>
 */
public final class SortByState implements SortStrategy {

    /**
     * Provides an explicit rank for each {@link TaskState}.
     * Null values are ranked last for safety.
     *
     * @param state the task state
     * @return integer rank (lower values come first)
     */
    private static int rank(TaskState state) {
        if (state == null) {
            return Integer.MAX_VALUE;
        }
        return switch (state) {
            case ToDo -> 0;
            case InProgress -> 1;
            case Completed -> 2;
        };
    }

    /**
     * Comparator used to order tasks by state, then title, then id.
     */
    private static final Comparator<ITask> CMP =
            Comparator
                    .comparingInt((ITask t) -> rank(t.getState()))
                    .thenComparing(t -> safe(t.getTitle()).toLowerCase())
                    .thenComparingInt(ITask::getId);

    /**
     * Returns the human-readable display name of this sorting strategy.
     *
     * @return the display name for this strategy
     */
    @Override
    public String displayName() {
        return "State (ToDo→Completed)";
    }

    /**
     * Returns a new list of tasks sorted by state (ToDo &lt; InProgress &lt; Completed),
     * then by case-insensitive title, then by id.
     *
     * @param items the list of tasks to be sorted (must not be {@code null})
     * @return a new list of sorted tasks
     * @throws NullPointerException if {@code items} is {@code null}
     */
    @Override
    public List<ITask> sort(List<ITask> items) {
        Objects.requireNonNull(items, "items");
        List<ITask> copy = new ArrayList<>(items);
        copy.sort(CMP);
        return copy;
    }

    /**
     * Returns a safe, non-null string for comparison.
     *
     * @param s the string to check
     * @return the original string if non-null, or an empty string if {@code null}
     */
    private static String safe(String s) {
        return (s == null) ? "" : s;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\sort\SortByTitle.java

SortByTitle

```java
package taskmanagement.application.viewmodel.sort;

import taskmanagement.domain.ITask;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Objects;

/**
 * Sorting strategy that orders tasks by their title in case-insensitive
 * lexicographic order. If titles are identical, tasks are ordered by ID
 * to ensure a stable ordering.
 */
public final class SortByTitle implements SortStrategy {

    /**
     * Comparator that orders tasks by title (case-insensitive),
     * then by task ID as a tiebreaker.
     */
    private static final Comparator<ITask> CMP =
            Comparator
                    .comparing((ITask t) -> safe(t.getTitle()).toLowerCase())
                    .thenComparingInt(ITask::getId);

    /**
     * Returns the human-readable display name of this sorting strategy.
     *
     * @return the display name for this strategy
     */
    @Override
    public String displayName() {
        return "Title (A→Z)";
    }

    /**
     * Returns a new list of tasks sorted by title (case-insensitive),
     * then by ID.
     *
     * @param items the list of tasks to sort (must not be {@code null})
     * @return a new list containing the tasks in sorted order
     * @throws NullPointerException if {@code items} is {@code null}
     */
    @Override
    public List<ITask> sort(List<ITask> items) {
        Objects.requireNonNull(items, "items");
        List<ITask> copy = new ArrayList<>(items);
        copy.sort(CMP);
        return copy;
    }

    /**
     * Returns a safe, non-null string for comparison.
     *
     * @param s the string to check
     * @return the original string if non-null, otherwise an empty string
     */
    private static String safe(String s) {
        return (s == null) ? "" : s;
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\sort\SortByTitle.java

SortByTitle

```
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\sort\SortStrategy.java

SortStrategy

```java
package taskmanagement.application.viewmodel.sort;

import taskmanagement.domain.ITask;

import java.util.List;

/**
 * Defines a strategy interface for sorting collections of {@link ITask}.
 * <p>
 * Implementations determine the specific ordering (e.g., by ID, title, or state).
 * Strategies must be pure: the input list must not be mutated; instead,
 * a new sorted snapshot should be returned.
 * </p>
 */
public interface SortStrategy {

    /**
     * Returns a short, human-readable label for this strategy,
     * suitable for display in UI controls such as combo boxes or menus.
     *
     * @return the display name of the strategy
     */
    String displayName();

    /**
     * Produces a new list of tasks sorted according to this strategy.
     * The input list must not be modified.
     *
     * @param items the source list of tasks (must not be {@code null})
     * @return a new list containing the tasks in sorted order
     * @throws NullPointerException if {@code items} is {@code null}
     */
    List<ITask> sort(List<ITask> items);
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\TasksViewModel.java

TasksViewModel

```java
package taskmanagement.application.viewmodel;


import taskmanagement.application.viewmodel.commands.AddTaskCommand;
import taskmanagement.application.viewmodel.commands.CommandException;
import taskmanagement.application.viewmodel.commands.CommandStack;
import taskmanagement.application.viewmodel.commands.DeleteTaskCommand;
import taskmanagement.application.viewmodel.commands.MarkStateCommand;
import taskmanagement.application.viewmodel.commands.UpdateTaskCommand;
import taskmanagement.application.viewmodel.events.Property;
import taskmanagement.application.viewmodel.sort.SortStrategy;
import taskmanagement.domain.ITask;
import taskmanagement.domain.Task;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.filter.ITaskFilter;
import taskmanagement.domain.visitor.TaskVisitor;
import taskmanagement.domain.visitor.adapters.ByStateCsvExporter;
import taskmanagement.domain.visitor.adapters.ByStatePlainTextExporter;
import taskmanagement.domain.visitor.adapters.IReportExporter;
import taskmanagement.domain.visitor.export.CompletedTaskRec;
import taskmanagement.domain.visitor.export.InProgressTaskRec;
import taskmanagement.domain.visitor.export.ToDoTaskRec;
import taskmanagement.domain.visitor.export.CsvFlatTaskVisitor;
import taskmanagement.domain.visitor.export.PlainTextFlatTaskVisitor;
import taskmanagement.domain.visitor.reports.ByStateCount;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.*;

/**
 * ViewModel layer (MVVM) for managing tasks.
 * <p>
 * Responsibilities:
 * <ul>
 *   <li>Mediates between the Swing UI and the DAO-based model.</li>
 *   <li>Exposes immutable UI-friendly row DTOs.</li>
 *   <li>Provides CRUD and state-changing operations via the Command pattern with undo/redo.</li>
 *   <li>Supports filtering via Combinator and reporting/export via Visitor (records & pattern matching).</li>
 *   <li>Notifies the UI through observable properties (Observer pattern).</li>
 * </ul>
 */
public final class TasksViewModel {

    private final ITasksDAO dao;
    private final CommandStack commands = new CommandStack();
    private List<ITask> snapshot = Collections.emptyList();
    private volatile Optional<ITaskFilter> activeFilter = Optional.empty();
    private volatile Optional<SortStrategy> currentSort = Optional.empty();

    private final Property<List<RowDTO>> rowsProperty = new Property<>(List.of());
    private final Property<List<RowDTO>> filteredRowsProperty = new Property<>(List.of());
    private final Property<Boolean> canUndoProperty = new Property<>(false);
    private final Property<Boolean> canRedoProperty = new Property<>(false);
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\application\viewmodel\TasksViewModel.java

TasksViewModel

```java
    /**
     * Immutable UI row data transfer object.
     *
     * @param id          task id
     * @param title       task title
     * @param description task description
     * @param state       task state as string
     */
    public record RowDTO(int id, String title, String description, String state) { }

    /**
     * Constructs a new {@code TasksViewModel}.
     *
     * @param dao the tasks DAO (must not be {@code null})
     * @throws NullPointerException if {@code dao} is {@code null}
     */
    public TasksViewModel(ITasksDAO dao) {
        this.dao = Objects.requireNonNull(dao, "ITasksDAO must not be null");
        updateCommandAvailability();
    }

    // --- Observable properties (for UI binding) ---

    /** Property that emits unfiltered rows whenever data changes. */
    public Property<List<RowDTO>> rowsProperty() { return rowsProperty; }
    /** Property that emits filtered rows whenever data or filter changes. */
    public Property<List<RowDTO>> filteredRowsProperty() { return filteredRowsProperty; }
    /** Property that emits {@code true} when undo is available. */
    public Property<Boolean> canUndoProperty() { return canUndoProperty; }
    /** Property that emits {@code true} when redo is available. */
    public Property<Boolean> canRedoProperty() { return canRedoProperty; }

    // --- Sorting API ---

    /** Sets the active sorting strategy. Passing {@code null} clears sorting. */
    public void setSortStrategy(SortStrategy strategy) {
        this.currentSort = Optional.ofNullable(strategy);
        rowsProperty.setValue(buildAllRowsSorted());
        filteredRowsProperty.setValue(buildFilteredRowsSorted());
    }

    /** Clears any active sort strategy and reverts to DAO order. */
    public void clearSortStrategy() { setSortStrategy(null); }

    // --- Data reload and queries ---

    /**
     * Reloads tasks from the DAO into an internal snapshot
     * and updates observable properties.
     *
     * @throws TasksDAOException if DAO retrieval fails
     */
    public void reload() throws TasksDAOException {
        ITask[] arr = dao.getTasks();
        this.snapshot = (arr == null) ? List.of() : Arrays.asList(arr);

        rowsProperty.setValue(buildAllRowsSorted());
        filteredRowsProperty.setValue(buildFilteredRowsSorted());
        updateCommandAvailability();
```

TasksViewModel

```java
    }

    /** Returns a UI-friendly immutable snapshot of all tasks. */
    public List<RowDTO> getRows() {
        List<RowDTO> out = new ArrayList<>(snapshot.size());
        for (ITask t : snapshot) {
            out.add(new RowDTO(
                    t.getId(),
                    safe(t.getTitle()),
                    safe(t.getDescription()),
                    (t.getState() == null) ? "" : t.getState().name()
            ));
        }
        return Collections.unmodifiableList(out);
    }

    /** Finds a row by its id. */
    public Optional<RowDTO> findRowById(int id) {
        validateId(id);
        for (ITask t : snapshot) if (t.getId() == id) return Optional.of(toRowDTO(t));
        return Optional.empty();
    }

    /** Returns available task states for UI binding. */
    public TaskState[] getAvailableStates() { return TaskState.values(); }

    // --- Filtering API (Combinator) ---

    /** Sets the active filter and updates filtered rows. */
    public void setFilter(ITaskFilter filter) {
        this.activeFilter = Optional.of(Objects.requireNonNull(filter, "filter must not be null"));
        filteredRowsProperty.setValue(buildFilteredRowsSorted());
    }

    /** Clears the active filter and updates filtered rows. */
    public void clearFilter() {
        this.activeFilter = Optional.empty();
        filteredRowsProperty.setValue(buildFilteredRowsSorted());
    }

    /** Returns a snapshot of rows filtered with the active filter (if any). */
    public List<RowDTO> getFilteredRows() {
        final List<RowDTO> out = new ArrayList<>();
        final var opt = this.activeFilter;

        if (opt.isEmpty()) {
            for (ITask t : snapshot) out.add(toRowDTO(t));
        } else {
            final ITaskFilter f = opt.get();
            for (ITask t : snapshot) if (f.test(t)) out.add(toRowDTO(t));
        }
        return Collections.unmodifiableList(out);
    }

    // --- Command operations (CRUD + State) ---

    /** Adds a new task via {@link AddTaskCommand}. */
    public void addTask(String title, String description, TaskState state) throws CommandException {
        validateTitle(title);
        Objects.requireNonNull(state, "state must not be null");

        Task toAdd = new Task(0, title, description, state);
        commands.execute(new AddTaskCommand(dao, toAdd));
        refreshAfterMutation();
    }

    /** Updates an existing task. */
    public void updateTask(int id, String title, String description, TaskState state) throws CommandException {
        validateId(id);
        validateTitle(title);
        Objects.requireNonNull(state, "state must not be null");

        final ITask before;
        try {
            before = dao.getTask(id);
        } catch (TasksDAOException e) {
            throw new CommandException("Failed to load task for update: id=" + id, e);
        }
        if (before == null) throw new CommandException("Task not found for update: id=" + id);

        Task after = new Task(id, title, description, state);
        commands.execute(new UpdateTaskCommand(dao, before, after));
        refreshAfterMutation();
    }

    /** Deletes a task. */
    public void deleteTask(int id) throws CommandException {
        validateId(id);
        final ITask snapshotToDelete;
        try {
            snapshotToDelete = dao.getTask(id);
        } catch (TasksDAOException e) {
            throw new CommandException("Failed to load task for delete: id=" + id, e);
        }
        if (snapshotToDelete == null) throw new CommandException("Task not found for delete: id=" + id);
        commands.execute(new DeleteTaskCommand(dao, snapshotToDelete));
        refreshAfterMutation();
    }

    /** Deletes multiple tasks. */
    public void deleteTasks(Collection<Integer> ids) throws CommandException {
        if (ids == null || ids.isEmpty()) return;
        boolean changed = false;

        for (Integer id : ids) {
            if (id == null) continue;
            validateId(id);
            final ITask t;
            try {
                t = dao.getTask(id);
            } catch (TasksDAOException e) {
                if (isNotFound(e)) continue;
                throw new CommandException("Failed to load task for delete: id=" + id, e);
            }
            if (t != null) {
                commands.execute(new DeleteTaskCommand(dao, t));
                changed = true;
            }
        }
        if (changed) refreshAfterMutation();
    }

    /** Deletes multiple tasks (varargs). */
    public void deleteTasks(int... ids) throws CommandException {
        if (ids == null || ids.length == 0) return;
        List<Integer> list = new ArrayList<>(ids.length);
        for (int id : ids) list.add(id);
        deleteTasks(list);
    }

    /** Deletes all tasks. */
    public void deleteAll() throws TasksDAOException {
        dao.deleteTasks();
        reload();
    }

    /** Marks a task with an explicit state. */
    public void markState(int id, TaskState state) throws CommandException {
        transitionState(id, state);
    }

    /**
     * Transitions a task to a new state if allowed.
     *
     * @param id     task id
     * @param target target state
     * @throws CommandException if task not found, illegal transition, or DAO fails
     */
    public void transitionState(int id, TaskState target) throws CommandException {
        validateId(id);
        Objects.requireNonNull(target, "target state must not be null");

        final ITask before;
        try {
            before = dao.getTask(id);
        } catch (TasksDAOException e) {
            throw new CommandException("Failed to load task for transition: id=" + id, e);
        }
        if (before == null) throw new CommandException("Task not found for transition: id=" + id);

        final TaskState current = before.getState();
        if (current == null) throw new CommandException("Task has no current state: id=" + id);

        int curIdx = stateIndex(current);
        int tgtIdx = stateIndex(target);
        if (tgtIdx < curIdx) {
            throw new CommandException("Backward transition not allowed: " + current + " -> " + target);
        }
        if (tgtIdx == curIdx) {
            return; // no-op
        }

        // MarkStateCommand expects (dao, before, targetState, TaskFactory)
        MarkStateCommand.TaskFactory factory = (prev, state) ->
                new Task(prev.getId(), prev.getTitle(), prev.getDescription(), state);

        commands.execute(new MarkStateCommand(dao, before, target, factory));
        refreshAfterMutation();
    }

    /**
     * Advances a task to its next legal state.
     *
     * @param id task id
     * @throws CommandException if task not found or DAO fails
     */
    public void advanceState(int id) throws CommandException {
        validateId(id);
        final ITask before;
        try {
            before = dao.getTask(id);
        } catch (TasksDAOException e) {
            throw new CommandException("Failed to load task for advance: id=" + id, e);
        }
        if (before == null) throw new CommandException("Task not found for advance: id=" + id);

        TaskState cur = before.getState();
        if (cur == null) throw new CommandException("Task has no current state: id=" + id);

        final TaskState next = switch (cur) {
            case ToDo       -> TaskState.InProgress;
            case InProgress -> TaskState.Completed;
            case Completed  -> null;
        };
        if (next == null) return; // already terminal

        MarkStateCommand.TaskFactory factory = (prev, state) ->
                new Task(prev.getId(), prev.getTitle(), prev.getDescription(), state);

        commands.execute(new MarkStateCommand(dao, before, next, factory));
        refreshAfterMutation();
    }


    // --- Undo/Redo ---

    /** @return true if undo is possible */
    public boolean canUndo() { return commands.canUndo(); }
    /** @return true if redo is possible */
    public boolean canRedo() { return commands.canRedo(); }

    /** Undoes the last executed command. */
    public void undo() throws CommandException {
        commands.undo();
        refreshAfterMutation();
    }

    /** Redoes the last undone command. */
    public void redo() throws CommandException {
        commands.redo();
        refreshAfterMutation();
    }

    // --- Reporting & Export ---

    /**
```

```java
     * Returns a report of task counts by state.
     *
     * @param useFiltered true to apply filter
     * @return counts by state
     */
    public ByStateCount getCountsByState(boolean useFiltered) {
        final ByStateCount out = new ByStateCount();
        final List<ITask> base = useFiltered ? extractFilteredTasks() : (snapshot == null ? List.of() : snapshot);
        for (ITask t : base) {
            if (t == null || t.getState() == null) continue;
            out.inc(t.getState());
        }
        return out;
    }

    /**
     * Returns counts as a map by state.
     *
     * @param useFiltered true to apply filter
     * @return map of counts
     */
    public EnumMap<TaskState, Integer> getCountsMapByState(boolean useFiltered) {
        final ByStateCount c = getCountsByState(useFiltered);
        final EnumMap<TaskState, Integer> map = new EnumMap<>(TaskState.class);
        map.put(TaskState.ToDo,       c.todo());
        map.put(TaskState.InProgress, c.inProgress());
        map.put(TaskState.Completed,  c.completed());
        return map;
    }

    /**
     * Exports tasks to a file (flat list).
     *
     * @param path        file path
     * @param format      export format
     * @param useFiltered whether to apply filter
     * @throws IOException       if IO fails
     * @throws TasksDAOException if DAO fails
     */
    public void exportTasks(Path path, ExportFormat format, boolean useFiltered)
            throws IOException, TasksDAOException {
        exportTasks(path, format, useFiltered, /*filteredIds*/ null);
    }

    /**
     * Exports tasks with optional filtered IDs (flat list).
     *
     * @param path        file path
     * @param format      export format
     * @param useFiltered whether to apply filter
     * @param filteredIds optional filtered ids (export only these if provided)
     * @throws IOException       if IO fails
     * @throws TasksDAOException if DAO fails
     */
    public void exportTasks(Path path, ExportFormat format, boolean useFiltered, List<Integer> filteredIds)
            throws IOException, TasksDAOException {
        Objects.requireNonNull(path, "path");
        Objects.requireNonNull(format, "format");
        final List<ITask> tasks = loadTasksForExport(useFiltered, filteredIds);

        switch (format) {
            case CSV -> {
                var visitor = new CsvFlatTaskVisitor();
                for (ITask t : tasks) {
                    if (t == null || t.getState() == null) continue;
                    String title = safe(t.getTitle());
                    String desc  = safe(t.getDescription());
                    switch (t.getState()) {
                        case ToDo       -> visitor.visit(new ToDoTaskRec(t.getId(), title, desc));
                        case InProgress -> visitor.visit(new InProgressTaskRec(t.getId(), title, desc));
                        case Completed  -> visitor.visit(new CompletedTaskRec(t.getId(), title, desc));
                    }
                }
                visitor.complete();
                String content = visitor.result();
                if (path.getParent() != null) Files.createDirectories(path.getParent());
                Files.writeString(path, content, StandardCharsets.UTF_8);
            }
            case TXT -> {
                var visitor = new PlainTextFlatTaskVisitor();
                for (ITask t : tasks) {
                    if (t == null || t.getState() == null) continue;
                    String title = safe(t.getTitle());
                    String desc  = safe(t.getDescription());
                    switch (t.getState()) {
                        case ToDo       -> visitor.visit(new ToDoTaskRec(t.getId(), title, desc));
                        case InProgress -> visitor.visit(new InProgressTaskRec(t.getId(), title, desc));
                        case Completed  -> visitor.visit(new CompletedTaskRec(t.getId(), title, desc));
                    }
                }
                visitor.complete();
                String content = visitor.result();
                if (path.getParent() != null) Files.createDirectories(path.getParent());
                Files.writeString(path, content, StandardCharsets.UTF_8);
            }
        }
    }



    /**
     * Exports a by-state report (aggregated counts).
     *
     * @param path        file path
     * @param format      export format
     * @param useFiltered whether to apply filter
     * @param filteredIds optional ids (ignored for aggregated unless you want to restrict the base set)
     * @throws IOException       if IO fails
     * @throws TasksDAOException if DAO fails
     */
    public void exportByStateReport(Path path, ExportFormat format, boolean useFiltered, List<Integer> filteredIds)
            throws IOException, TasksDAOException {
        Objects.requireNonNull(path, "path");
        Objects.requireNonNull(format, "format");

        // Count
        final List<ITask> base = loadTasksForExport(useFiltered, filteredIds);
        final ByStateCount counts = new ByStateCount();
        for (ITask t : base) {
            if (t == null || t.getState() == null) continue;
            counts.inc(t.getState());
        }

        // Your IReportExporter<T>.export(T) takes only the report; we'll write the returned content to file
        final IReportExporter<ByStateCount> exporter = switch (format) {
            case CSV -> new ByStateCsvExporter();
            case TXT -> new ByStatePlainTextExporter();
        };

        final String content = exporter.export(counts); // <-- no Path here
        if (path.getParent() != null) Files.createDirectories(path.getParent());
        Files.writeString(path, content, StandardCharsets.UTF_8);
    }


    // --- Helpers ---

    private void refreshAfterMutation() throws CommandException {
        try {
            reload();
        } catch (TasksDAOException e) {
            throw new CommandException("Failed to refresh after mutation", e);
        }
    }

    private void updateCommandAvailability() {
        canUndoProperty.setValue(commands.canUndo());
        canRedoProperty.setValue(commands.canRedo());
    }

    private List<ITask> loadTasksForExport(boolean useFiltered, List<Integer> filteredIds) throws TasksDAOException {
        if (useFiltered && filteredIds != null && !filteredIds.isEmpty()) {
            final List<ITask> out = new ArrayList<>(filteredIds.size());
            for (Integer id : filteredIds) {
                if (id == null) continue;
                final ITask t = dao.getTask(id);
                if (t != null) out.add(t);
            }
            return out;
        }
        final ITask[] arr = dao.getTasks();
        return (arr == null) ? List.of() : Arrays.asList(arr);
    }

    private List<ITask> extractFilteredTasks() {
        if (activeFilter.isEmpty()) return (snapshot == null) ? List.of() : snapshot;
        final ITaskFilter f = activeFilter.get();
        final List<ITask> out = new ArrayList<>();
        for (ITask t : snapshot) if (f.test(t)) out.add(t);
        return out;
    }

    private static String safe(String s) { return (s == null) ? "" : s; }

    private static void validateTitle(String title) {
        if (title == null || title.trim().isEmpty()) {
            throw new IllegalArgumentException("title must not be empty");
        }
    }

    private static void validateId(int id) {
        if (id < 0) throw new IllegalArgumentException("invalid id: " + id);
    }

    private static RowDTO toRowDTO(ITask t) {
        return new RowDTO(
                t.getId(),
                safe(t.getTitle()),
                safe(t.getDescription()),
                (t.getState() == null) ? "" : t.getState().name()
        );
    }

    private List<RowDTO> buildAllRowsSorted() {
        final List<ITask> tasks = sortedSnapshot();
        final List<RowDTO> out = new ArrayList<>(tasks.size());
        for (ITask t : tasks) out.add(toRowDTO(t));
        return Collections.unmodifiableList(out);
    }

    private List<RowDTO> buildFilteredRowsSorted() {
        final List<ITask> filtered = extractFilteredTasks();
        final List<ITask> ordered = currentSort.map(s -> s.sort(filtered)).orElse(filtered);
        final List<RowDTO> out = new ArrayList<>(ordered.size());
        for (ITask t : ordered) out.add(toRowDTO(t));
        return Collections.unmodifiableList(out);
    }

    private List<ITask> sortedSnapshot() {
        final List<ITask> base = (snapshot == null) ? List.of() : snapshot;
        return currentSort.map(s -> s.sort(base)).orElse(base);
    }

    private static boolean isNotFound(TasksDAOException e) {
        String msg = e.getMessage();
        return msg != null && msg.toLowerCase(Locale.ROOT).contains("not found");
    }

    private static int stateIndex(TaskState s) {
        return switch (s) {
            case ToDo -> 0;
            case InProgress -> 1;
            case Completed -> 2;
        };
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\exceptions\ValidationException.java

ValidationException

```java
package taskmanagement.domain.exceptions;

/**
 * Exception thrown to indicate validation errors in the domain layer.
 * <p>
 * Used for invalid input values or illegal state transitions within tasks.
 * This exception is unchecked and extends {@link RuntimeException}.
 * </p>
 */
public class ValidationException extends RuntimeException {

    /**
     * Constructs a new {@code ValidationException} with the specified detail message.
     *
     * @param message the detail message
     */
    public ValidationException(String message) {
        super(message);
    }

    /**
     * Constructs a new {@code ValidationException} with the specified detail message
     * and cause.
     *
     * @param message the detail message
     * @param cause   the underlying cause (may be {@code null})
     */
    public ValidationException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\filter\Filters.java

Filters

```java
package taskmanagement.domain.filter;

import taskmanagement.domain.TaskState;

/**
 * Utility class providing ready-made task filters as building blocks
 * for Combinator composition.
 * <p>
 * All methods are null-safe. Strings are trimmed and compared
 * in lowercase where applicable.
 * </p>
 */
public final class Filters {

    private Filters() { }

    /**
     * Returns a filter that matches tasks whose title contains the given substring,
     * case-insensitive.
     *
     * @param needle the substring to search for (nullable, treated as empty)
     * @return a task filter matching by title
     */
    public static ITaskFilter titleContains(String needle) {
        final String n = safeLower(needle);
        return t -> {
            String title = t.getTitle();
            return title != null && title.toLowerCase().contains(n);
        };
    }

    /**
     * Returns a filter that matches tasks whose description contains the given substring,
     * case-insensitive.
     *
     * @param needle the substring to search for (nullable, treated as empty)
     * @return a task filter matching by description
     */
    public static ITaskFilter descriptionContains(String needle) {
        final String n = safeLower(needle);
        return t -> {
            String desc = t.getDescription();
            return desc != null && desc.toLowerCase().contains(n);
        };
    }

    /**
     * Returns a filter that matches tasks with the specified id.
     *
     * @param id the task id
     * @return a task filter matching by id
     */
    public static ITaskFilter idEquals(int id) {
        return t -> t.getId() == id;
    }

    /**
     * Returns a filter that matches tasks in the specified state.
     *
```

Filters

```java
     * @param state the target state (nullable)
     * @return a task filter matching by state
     */
    public static ITaskFilter stateIs(TaskState state) {
        return t -> t.getState() == state;
    }

    /**
     * Alias for {@link #stateIs(TaskState)}.
     *
     * @param state the target state (nullable)
     * @return a task filter matching by state
     */
    public static ITaskFilter byState(TaskState state) {
        return stateIs(state);
    }

    /**
     * Returns a filter that negates the given filter.
     *
     * @param f the filter to negate
     * @return negated filter
     */
    public static ITaskFilter not(ITaskFilter f) {
        return ITaskFilter.not(f);
    }

    /**
     * Returns a filter that matches all tasks.
     *
     * @return match-all filter
     */
    public static ITaskFilter all() {
        return ITaskFilter.all();
    }

    private static String safeLower(String s) {
        return (s == null) ? "" : s.trim().toLowerCase();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\filter\ITaskFilter.java

ITaskFilter

```java
package taskmanagement.domain.filter;

import taskmanagement.domain.ITask;

/**
 * Functional filter over tasks implementing the Combinator pattern.
 * <p>
 * Filters can be combined using logical operators:
 * <ul>
 *   <li>{@link #and(ITaskFilter)} - logical AND composition</li>
 *   <li>{@link #or(ITaskFilter)} - logical OR composition</li>
 *   <li>{@link #not(ITaskFilter)} - static logical NOT</li>
 *   <li>{@link #all()} - static match-all filter</li>
 * </ul>
 * </p>
 */
@FunctionalInterface
public interface ITaskFilter {

    /**
     * Tests whether a given task matches the filter.
     *
     * @param task the task to test (must not be {@code null})
     * @return {@code true} if the task matches, otherwise {@code false}
     */
    boolean test(ITask task);

    /**
     * Returns a composed filter representing the logical AND of this filter
     * and another filter.
     *
     * @param other the other filter (must not be {@code null})
     * @return composed filter that matches when both filters match
     */
    default ITaskFilter and(ITaskFilter other) {
        return t -> this.test(t) && other.test(t);
    }

    /**
     * Returns a composed filter representing the logical OR of this filter
     * and another filter.
     *
     * @param other the other filter (must not be {@code null})
     * @return composed filter that matches when either filter matches
     */
    default ITaskFilter or(ITaskFilter other) {
        return t -> this.test(t) || other.test(t);
    }

    /**
     * Returns a filter representing the logical NOT of the given filter.
     *
     * @param f the filter to negate (must not be {@code null})
     * @return filter that matches when the given filter does not match
     */
    static ITaskFilter not(ITaskFilter f) {
        return t -> !f.test(t);
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\filter\ITaskFilter.java

ITaskFilter

```java
    /**
     * Returns a filter that matches all tasks.
     *
     * @return match-all filter
     */
    static ITaskFilter all() {
        return t -> true;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\ITask.java

ITask

```java
package taskmanagement.domain;

import taskmanagement.domain.visitor.TaskVisitor;

/**
 * Represents a task entity in the system.
 * <p>
 * Tasks expose their identity, descriptive fields, lifecycle state,
 * and support visiting via the {@link TaskVisitor} interface.
 * </p>
 */
public interface ITask {

    /**
     * Returns the unique identifier of the task.
     *
     * @return task id
     */
    int getId();

    /**
     * Returns the title of the task.
     *
     * @return task title (may be {@code null})
     */
    String getTitle();

    /**
     * Returns the description of the task.
     *
     * @return task description (may be {@code null})
     */
    String getDescription();

    /**
     * Returns the current lifecycle state of the task.
     *
     * @return task state (never {@code null})
     */
    TaskState getState();

    /**
     * Accepts a {@link TaskVisitor}, allowing operations to be
     * performed on this task instance using the Visitor pattern.
     *
     * @param visitor the visitor to accept (must not be {@code null})
     */
    void accept(TaskVisitor visitor);
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\Task.java

Task

```java
package taskmanagement.domain;

import taskmanagement.domain.exceptions.ValidationException;
import taskmanagement.domain.visitor.TaskVisitor;
import taskmanagement.domain.visitor.export.CompletedTaskRec;
import taskmanagement.domain.visitor.export.InProgressTaskRec;
import taskmanagement.domain.visitor.export.ToDoTaskRec;

import java.util.Objects;

/**
 * Task entity with immutable identity (id), validated fields, and a behavioral state
 * managed by the {@link TaskState} state machine.
 * <p>
 * Constructors delegate to setters to reuse validation logic. State changes are validated
 * to allow only legal forward transitions, and visitor support is provided via
 * {@link #accept(TaskVisitor)}.
 * </p>
 */
public class Task implements ITask {

    private int id;
    private String title;
    private String description;
    private TaskState state;

    /**
     * Constructs a task with all fields; setters perform validation.
     *
     * @param id          task id (non-negative recommended; DAO may assign)
     * @param title       non-empty title
     * @param description nullable description
     * @param state       non-null state
     * @throws IllegalArgumentException if {@code title} is blank
     * @throws ValidationException      if {@code state} is {@code null} or transition is illegal
     */
    public Task(int id, String title, String description, TaskState state) {
        setId(id);
        setTitle(title);
        setDescription(description);
        setState(state);
    }

    /**
     * Copy constructor.
     *
     * @param other another task instance to copy (must not be {@code null})
     * @throws NullPointerException if {@code other} is {@code null}
     */
    public Task(Task other) {
        this(Objects.requireNonNull(other, "other must not be null").id,
                other.title, other.description, other.state);
    }

    // ------------------------------------------------------------
    // ITask
    // ------------------------------------------------------------

    /** {@inheritDoc} */
    @Override public int getId() { return id; }

    /**
     * Sets the identifier.
     * <p>
     * Negative ids are tolerated only if your DAO uses them as placeholders;
     * otherwise prefer non-negative ids.
     * </p>
     *
     * @param id the identifier to set
     */
    public void setId(int id) { this.id = id; }

    /** {@inheritDoc} */
    @Override public String getTitle() { return title; }

    /**
     * Sets a non-blank title.
     *
     * @param title the title to set (must not be {@code null} or blank)
     * @throws IllegalArgumentException if {@code title} is {@code null} or blank
     */
    public void setTitle(String title) {
        if (title == null || title.trim().isEmpty()) {
            throw new IllegalArgumentException("title must not be empty");
        }
        this.title = title.trim();
    }

    /** {@inheritDoc} */
    @Override public String getDescription() { return description; }

    /**
     * Sets the description (nullable). Trims when non-null.
     *
     * @param description description text, or {@code null}
     */
    public void setDescription(String description) {
        this.description = (description == null) ? null : description.trim();
    }

    /** {@inheritDoc} */
    @Override public TaskState getState() { return state; }

    /**
     * Sets the state and enforces forward-only transitions.
     * <p>
     * Allowed transitions: ToDo → InProgress → Completed. Setting the same state is
     * idempotent. Throws unchecked {@link ValidationException} to keep caller APIs simple.
     * </p>
     *
     * @param newState the new state (must not be {@code null})
     * @throws ValidationException if {@code newState} is {@code null} or the transition is illegal
     */
    public void setState(TaskState newState) {
        if (newState == null) {
            throw new ValidationException("state must not be null");
        }
        if (this.state == newState) {
```

Task

```java
            return; // idempotent
        }
        if (this.state != null && !this.state.canTransitionTo(newState)) {
            throw new ValidationException("Illegal state transition: " + this.state + " -> " + newState);
        }
        this.state = newState;
    }

    /**
     * Transitions to a target state if allowed by the current state's rules.
     *
     * @param target desired target state (must not be {@code null})
     * @throws ValidationException  if the transition is not allowed
     * @throws NullPointerException if {@code target} is {@code null}
     */
    public void transitionTo(TaskState target) throws ValidationException {
        Objects.requireNonNull(target, "target must not be null");
        if (this.state == null || this.state.canTransitionTo(target)) {
            this.state = target;
        } else {
            throw new ValidationException("Illegal state transition: " + this.state + " -> " + target);
        }
    }

    /**
     * Advances to the next state according to the current state's behavior.
     *
     * @throws ValidationException if advancing is not allowed
     */
    public void advanceState() throws ValidationException {
        transitionTo(Objects.requireNonNull(this.state, "current state is null").next());
    }

    // ------------------------------------------------------------
    // Visitor
    // ------------------------------------------------------------

    /**
     * Accepts a {@link TaskVisitor} and dispatches to the record-typed visit
     * based on the current {@link TaskState}.
     *
     * @param visitor the visitor to accept (must not be {@code null})
     * @throws NullPointerException if {@code visitor} or {@code state} is {@code null}
     */
    @Override
    public void accept(TaskVisitor visitor) {
        Objects.requireNonNull(visitor, "visitor must not be null");
        final TaskState s = Objects.requireNonNull(this.state, "state must not be null");
        switch (s) {
            case ToDo       -> visitor.visit(new ToDoTaskRec(id, title, description));
            case InProgress -> visitor.visit(new InProgressTaskRec(id, title, description));
            case Completed  -> visitor.visit(new CompletedTaskRec(id, title, description));
        }
    }

    // ------------------------------------------------------------
    // Object contracts
    // ------------------------------------------------------------
```

Task

```java
    /**
     * Compares tasks by identity (id) only.
     *
     * @param o other object
     * @return {@code true} if the other object is a {@code Task} with the same id
     */
    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Task other)) return false;
        return id == other.id;
    }

    /** {@inheritDoc} */
    @Override public int hashCode() { return Integer.hashCode(id); }

    /**
     * Returns a string representation of the task.
     *
     * @return string form including id, title, description, and state
     */
    @Override
    public String toString() {
        return "Task{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", description='" + description + '\'' +
                ", state=" + state +
                '}';
    }
}
```

TaskState

```java
package taskmanagement.domain;

/**
 * Task lifecycle states implemented as a behavioral enum (State pattern).
 * <p>
 * Each constant defines its own transition rules and "next" step.
 * Allowed transitions:
 * <ul>
 *   <li>ToDo → InProgress or ToDo (idempotent)</li>
 *   <li>InProgress → Completed or InProgress (idempotent)</li>
 *   <li>Completed → Completed (idempotent only)</li>
 * </ul>
 * </p>
 */
public enum TaskState {

    /**
     * Initial state for newly created tasks.
     */
    ToDo {
        /** {@inheritDoc} */
        @Override
        public boolean canTransitionTo(TaskState target) {
            return target == ToDo || target == InProgress;
        }

        /** {@inheritDoc} */
        @Override
        public TaskState next() {
            return InProgress;
        }
    },

    /**
     * Active work state.
     * Rollback to {@link #ToDo} is not allowed.
     */
    InProgress {
        /** {@inheritDoc} */
        @Override
        public boolean canTransitionTo(TaskState target) {
            return target == InProgress || target == Completed;
        }

        /** {@inheritDoc} */
        @Override
        public TaskState next() {
            return Completed;
        }
    },

    /**
     * Terminal state for finished tasks.
     */
    Completed {
        /** {@inheritDoc} */
        @Override
        public boolean canTransitionTo(TaskState target) {
            return target == Completed;
```

TaskState

```java
        }

        /** {@inheritDoc} */
        @Override
        public TaskState next() {
            return Completed;
        }
    };

    /**
     * Checks if a transition from this state to the given target state is allowed.
     *
     * @param target the target state
     * @return {@code true} if the transition is allowed, otherwise {@code false}
     */
    public abstract boolean canTransitionTo(TaskState target);

    /**
     * Returns the "next" state in the lifecycle.
     * <p>
     * For terminal states, returns itself.
     * </p>
     *
     * @return the next state
     */
    public abstract TaskState next();
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\adapters\ByStateCsvExporter.java

ByStateCsvExporter

```java
package taskmanagement.domain.visitor.adapters;

import taskmanagement.domain.visitor.reports.ByStateCount;

/**
 * CSV exporter for {@link ByStateCount} reports.
 * <p>
 * Output format:
 * <pre>
 * state,count
 * ToDo,&lt;n&gt;
 * InProgress,&lt;n&gt;
 * Completed,&lt;n&gt;
 * Total,&lt;n&gt;
 * </pre>
 * </p>
 */
public final class ByStateCsvExporter implements IReportExporter<ByStateCount> {

    /**
     * Exports a {@link ByStateCount} report to CSV format.
     *
     * @param report the report to export (must not be {@code null})
     * @return CSV string representation of the report
     * @throws IllegalArgumentException if {@code report} is {@code null}
     */
    @Override
    public String export(ByStateCount report) {
        if (report == null) {
            throw new IllegalArgumentException("report is null");
        }
        int total = report.todo() + report.inProgress() + report.completed();

        String nl = System.lineSeparator();
        StringBuilder sb = new StringBuilder();
        sb.append("state,count").append(nl);
        sb.append("ToDo,").append(report.todo()).append(nl);
        sb.append("InProgress,").append(report.inProgress()).append(nl);
        sb.append("Completed,").append(report.completed()).append(nl);
        sb.append("Total,").append(total).append(nl);
        return sb.toString();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\adapters\ByStatePlainTextExporter.java

ByStatePlainTextExporter

```java
package taskmanagement.domain.visitor.adapters;

import taskmanagement.domain.visitor.reports.ByStateCount;

/**
 * Plain text exporter for {@link ByStateCount} reports.
 * <p>
 * Output format:
 * <pre>
 * Tasks by state
 * ToDo: &lt;n&gt;
 * InProgress: &lt;n&gt;
 * Completed: &lt;n&gt;
 * Total: &lt;n&gt;
 * </pre>
 * </p>
 */
public final class ByStatePlainTextExporter implements IReportExporter<ByStateCount> {

    /**
     * Exports a {@link ByStateCount} report to plain text format.
     *
     * @param report the report to export (must not be {@code null})
     * @return plain text string representation of the report
     * @throws IllegalArgumentException if {@code report} is {@code null}
     */
    @Override
    public String export(ByStateCount report) {
        if (report == null) {
            throw new IllegalArgumentException("report is null");
        }
        int total = report.todo() + report.inProgress() + report.completed();

        String nl = System.lineSeparator();
        StringBuilder sb = new StringBuilder();
        sb.append("Tasks by state").append(nl);
        sb.append("ToDo: ").append(report.todo()).append(nl);
        sb.append("InProgress: ").append(report.inProgress()).append(nl);
        sb.append("Completed: ").append(report.completed()).append(nl);
        sb.append("Total: ").append(total).append(nl);
        return sb.toString();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\adapters\IReportExporter.java

IReportExporter

```java
package taskmanagement.domain.visitor.adapters;

import taskmanagement.domain.visitor.reports.Report;

/**
 * Adapter interface for exporting a {@link Report} into a textual representation.
 *
 * @param <T> the specific type of {@link Report} supported by this exporter
 */
public interface IReportExporter<T extends Report> {

    /**
     * Exports the given report into a textual format.
     *
     * @param report the report to export (must not be {@code null})
     * @return string content representing the exported report
     * @throws IllegalArgumentException if {@code report} is {@code null}
     *                                  or if the exporter does not support the report type
     */
    String export(T report);
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\CountByStateVisitor.java

CountByStateVisitor

```java
package taskmanagement.domain.visitor;

import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.visitor.export.CompletedTaskRec;
import taskmanagement.domain.visitor.export.InProgressTaskRec;
import taskmanagement.domain.visitor.export.ToDoTaskRec;
import taskmanagement.domain.visitor.reports.ByStateCount;

/**
 * Visitor that counts tasks by their {@link TaskState} and produces
 * a {@link ByStateCount} report.
 * <p>
 * Supports record-based visits (Visitor + Records + Pattern Matching).
 * Can also be used directly with {@link ITask} or {@link TaskState}.
 * </p>
 */
public final class CountByStateVisitor implements TaskVisitor {

    private int todo;
    private int inProgress;
    private int completed;

    /**
     * Resets all counters to zero.
     * Useful when reusing the same visitor instance.
     */
    public void reset() {
        todo = inProgress = completed = 0;
    }

    /**
     * Counts a task by delegating to its {@link ITask#accept(TaskVisitor)} method.
     *
     * @param task the task to count (nullable; ignored if {@code null})
     */
    public void visit(ITask task) {
        if (task != null) {
            task.accept(this);
        }
    }

    /**
     * Increments counters directly based on a {@link TaskState}.
     *
     * @param s the task state (nullable; ignored if {@code null})
     */
    public void visit(TaskState s) {
        if (s == null) return;
        switch (s) {
            case ToDo -> todo++;
            case InProgress -> inProgress++;
            case Completed -> completed++;
        }
    }

    @Override
    public void visit(ToDoTaskRec node) {
        todo++;
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\CountByStateVisitor.java

CountByStateVisitor

```java
    }

    @Override
    public void visit(InProgressTaskRec node) {
        inProgress++;
    }

    @Override
    public void visit(CompletedTaskRec node) {
        completed++;
    }

    @Override
    public void complete() {
        // no-op, placeholder for future extensions
    }

    /**
     * Returns the current counts as a {@link ByStateCount} report.
     *
     * @return report with counts for ToDo, InProgress, and Completed states
     */
    public ByStateCount result() {
        return new ByStateCount(todo, inProgress, completed);
    }

    /**
     * Alias for {@link #result()}.
     *
     * @return report with counts for ToDo, InProgress, and Completed states
     */
    public ByStateCount report() {
        return result();
    }

    /**
     * Alias for {@link #result()}.
     *
     * @return report with counts for ToDo, InProgress, and Completed states
     */
    public ByStateCount getReport() {
        return result();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\CompletedTaskRec.java

CompletedTaskRec

```java
package taskmanagement.domain.visitor.export;

import taskmanagement.domain.TaskState;

/**
 * Export record representing a task in the {@link TaskState#Completed} state.
 * <p>
 * Used by export visitors to handle tasks with state {@code Completed}.
 * </p>
 *
 * @param id          the task id
 * @param title       the task title
 * @param description the task description
 */
public record CompletedTaskRec(int id, String title, String description)
        implements ExportNode {

    /**
     * Returns the associated state for this export record.
     *
     * @return {@link TaskState#Completed}
     */
    @Override
    public TaskState state() {
        return TaskState.Completed;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\CsvFlatTaskVisitor.java

CsvFlatTaskVisitor

```java
package taskmanagement.domain.visitor.export;

import taskmanagement.domain.visitor.TaskVisitor;

/**
 * Visitor implementation that produces CSV (UTF-8) text output
 * for tasks by visiting {@link ExportNode} variants.
 * <p>
 * Output format:
 * <pre>
 * id,title,description,state
 * 1,"Task A","Description A",ToDo
 * 2,"Task B","Description B",Completed
 * ...
 * </pre>
 * </p>
 */
public final class CsvFlatTaskVisitor implements TaskVisitor {

    private final StringBuilder sb = new StringBuilder("id,title,description,state\n");

    /**
     * Escapes a string for inclusion in CSV by surrounding with quotes
     * and doubling internal quotes.
     *
     * @param s the input string (nullable)
     * @return the escaped string, never {@code null}
     */
    private static String esc(String s) {
        if (s == null) return "";
        return "\"" + s.replace("\"", "\"\"") + "\"";
    }

    /**
     * Appends a row for the given export node.
     *
     * @param n the export node (must not be {@code null})
     */
    private void addRow(ExportNode n) {
        sb.append(n.id()).append(',')
                .append(esc(n.title())).append(',')
                .append(esc(n.description())).append(',')
                .append(n.state().name())
                .append('\n');
    }

    @Override
    public void visit(ToDoTaskRec node) {
        addRow(node);
    }

    @Override
    public void visit(InProgressTaskRec node) {
        addRow(node);
    }

    @Override
    public void visit(CompletedTaskRec node) {
        addRow(node);
```

CsvFlatTaskVisitor

```java
    }

    /**
     * Returns the accumulated CSV content.
     *
     * @return CSV text
     */
    public String result() {
        return sb.toString();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\ExportNode.java

ExportNode

```java
package taskmanagement.domain.visitor.export;

import taskmanagement.domain.TaskState;

/**
 * Sealed interface representing exportable task nodes for use in visitors.
 * <p>
 * Only the record variants {@link ToDoTaskRec}, {@link InProgressTaskRec},
 * and {@link CompletedTaskRec} are permitted implementations.
 * </p>
 */
public sealed interface ExportNode
        permits ToDoTaskRec, InProgressTaskRec, CompletedTaskRec {

    /**
     * Returns the unique identifier of the task.
     *
     * @return task id
     */
    int id();

    /**
     * Returns the title of the task.
     *
     * @return task title (may be {@code null})
     */
    String title();

    /**
     * Returns the description of the task.
     *
     * @return task description (may be {@code null})
     */
    String description();

    /**
     * Returns the state of the task.
     *
     * @return task state (never {@code null})
     */
    TaskState state();
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\InProgressTaskRec.java

InProgressTaskRec

```java
package taskmanagement.domain.visitor.export;

import taskmanagement.domain.TaskState;

/**
 * Export record representing a task in the {@link TaskState#InProgress} state.
 * <p>
 * Used by export visitors to handle tasks with state {@code InProgress}.
 * </p>
 *
 * @param id          the task id
 * @param title       the task title
 * @param description the task description
 */
public record InProgressTaskRec(int id, String title, String description)
        implements ExportNode {

    /**
     * Returns the associated state for this export record.
     *
     * @return {@link TaskState#InProgress}
     */
    @Override
    public TaskState state() {
        return TaskState.InProgress;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\PlainTextFlatTaskVisitor.java

PlainTextFlatTaskVisitor

```java
package taskmanagement.domain.visitor.export;

import taskmanagement.domain.visitor.TaskVisitor;

/**
 * Visitor implementation that produces plain-text block output
 * for tasks by visiting {@link ExportNode} variants.
 * <p>
 * Output format example:
 * <pre>
 * Tasks Export
 * ------------
 * ID: 1
 * Title: Task A
 * Description: Some description
 * State: ToDo
 *
 * ID: 2
 * Title: Task B
 * Description: Another description
 * State: Completed
 * </pre>
 * </p>
 */
public final class PlainTextFlatTaskVisitor implements TaskVisitor {

    private final StringBuilder sb = new StringBuilder("Tasks Export\n------------\n");

    /**
     * Returns a non-null string (empty if {@code s} is {@code null}).
     *
     * @param s input string (nullable)
     * @return non-null string
     */
    private static String nz(String s) {
        return s == null ? "" : s;
    }

    /**
     * Appends a formatted block for the given export node.
     *
     * @param n the export node (must not be {@code null})
     */
    private void addBlock(ExportNode n) {
        sb.append("ID: ").append(n.id()).append('\n')
                .append("Title: ").append(nz(n.title())).append('\n')
                .append("Description: ").append(nz(n.description())).append('\n')
                .append("State: ").append(n.state().name()).append("\n\n");
    }

    @Override
    public void visit(ToDoTaskRec node) {
        addBlock(node);
    }

    @Override
    public void visit(InProgressTaskRec node) {
        addBlock(node);
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\PlainTextFlatTaskVisitor.java

PlainTextFlatTaskVisitor

```java
    @Override
    public void visit(CompletedTaskRec node) {
        addBlock(node);
    }

    /**
     * Returns the accumulated plain-text output.
     *
     * @return plain-text export result
     */
    public String result() {
        return sb.toString();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\export\ToDoTaskRec.java

ToDoTaskRec

```java
package taskmanagement.domain.visitor.export;

import taskmanagement.domain.TaskState;

/**
 * Export record representing a task in the {@link TaskState#ToDo} state.
 * <p>
 * Used by export visitors to handle tasks with state {@code ToDo}.
 * </p>
 *
 * @param id          the task id
 * @param title       the task title
 * @param description the task description
 */
public record ToDoTaskRec(int id, String title, String description)
        implements ExportNode {

    /**
     * Returns the associated state for this export record.
     *
     * @return {@link TaskState#ToDo}
     */
    @Override
    public TaskState state() {
        return TaskState.ToDo;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\ITaskVisitor.java

ITaskVisitor

```java
package taskmanagement.domain.visitor;

import taskmanagement.domain.visitor.export.ToDoTaskRec;
import taskmanagement.domain.visitor.export.InProgressTaskRec;
import taskmanagement.domain.visitor.export.CompletedTaskRec;

/**
 * Base visitor interface over task export record variants.
 * <p>
 * Provides type-specific visit methods for each record implementation,
 * enabling pattern matching on task state. Includes an optional
 * {@link #complete()} hook for batch traversal completion.
 * </p>
 */
public interface ITaskVisitor {

    /**
     * Visits a To-Do task record node.
     *
     * @param node record representing a To-Do task snapshot
     */
    void visit(ToDoTaskRec node);

    /**
     * Visits an In-Progress task record node.
     *
     * @param node record representing an In-Progress task snapshot
     */
    void visit(InProgressTaskRec node);

    /**
     * Visits a Completed task record node.
     *
     * @param node record representing a Completed task snapshot
     */
    void visit(CompletedTaskRec node);

    /**
     * Optional hook invoked after a batch traversal completes.
     * Default implementation is a no-op.
     */
    default void complete() { }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\reports\ByStateCount.java

ByStateCount

```java
package taskmanagement.domain.visitor.reports;

import taskmanagement.domain.TaskState;

import java.util.EnumMap;
import java.util.Map;

/**
 * Report data structure that aggregates the number of tasks per {@link TaskState}.
 * <p>
 * Provides both generic access methods ({@link #count(TaskState)}, {@link #inc(TaskState)})
 * and convenience accessors for individual states ({@link #todo()}, {@link #inProgress()},
 * {@link #completed()}).
 * </p>
 */
public final class ByStateCount implements Report {

    private int todo;
    private int inProgress;
    private int completed;

    /**
     * Creates a new {@code ByStateCount} with all counters initialized to zero.
     */
    public ByStateCount() {
    }

    /**
     * Creates a new {@code ByStateCount} with explicit counts.
     *
     * @param todo       initial count of tasks in {@link TaskState#ToDo}
     * @param inProgress initial count of tasks in {@link TaskState#InProgress}
     * @param completed  initial count of tasks in {@link TaskState#Completed}
     */
    public ByStateCount(int todo, int inProgress, int completed) {
        this.todo = todo;
        this.inProgress = inProgress;
        this.completed = completed;
    }

    /**
     * Increments the counter corresponding to the given state.
     *
     * @param state the task state to increment
     */
    public void inc(TaskState state) {
        switch (state) {
            case ToDo -> todo++;
            case InProgress -> inProgress++;
            case Completed -> completed++;
        }
    }

    /**
     * Returns the count for the given state.
     *
     * @param state the task state to query
     * @return number of tasks recorded in the specified state
     */
```

ByStateCount

```java
    public int count(TaskState state) {
        return switch (state) {
            case ToDo -> todo;
            case InProgress -> inProgress;
            case Completed -> completed;
        };
    }

    /**
     * Returns the count of tasks in {@link TaskState#ToDo}.
     *
     * @return number of tasks in ToDo state
     */
    public int todo() {
        return todo;
    }

    /**
     * Returns the count of tasks in {@link TaskState#InProgress}.
     *
     * @return number of tasks in InProgress state
     */
    public int inProgress() {
        return inProgress;
    }

    /**
     * Returns the count of tasks in {@link TaskState#Completed}.
     *
     * @return number of tasks in Completed state
     */
    public int completed() {
        return completed;
    }

    /**
     * Returns an immutable map view of all state counts.
     *
     * @return a map of {@link TaskState} to their corresponding counts
     */
    public Map<TaskState, Integer> asMap() {
        EnumMap<TaskState, Integer> m = new EnumMap<>(TaskState.class);
        m.put(TaskState.ToDo, todo);
        m.put(TaskState.InProgress, inProgress);
        m.put(TaskState.Completed, completed);
        return Map.copyOf(m);
    }

    /**
     * Returns a string representation of the state counts.
     *
     * @return a formatted string showing counts for each state
     */
    @Override
    public String toString() {
        return "ByStateCount{ToDo=" + todo +
                ", InProgress=" + inProgress +
                ", Completed=" + completed + '}';
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\reports\ByStateCount.java

ByStateCount

```
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\reports\Report.java

Report

```java
package taskmanagement.domain.visitor.reports;

/**
 * Marker interface for all report result types produced by TaskVisitor
 * implementations in the Tasks Management Application.
 * <p>
 * Serves as a common supertype for specific report result classes such
 * as {@code ByStateCount}.
 * </p>
 */
public interface Report {
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\domain\visitor\TaskVisitor.java

TaskVisitor

```java
package taskmanagement.domain.visitor;

/**
 * Visitor interface required by the project (see Requirements.md).
 * <p>
 * Extends {@link ITaskVisitor} so that domain tasks can call
 * {@code ITask.accept(TaskVisitor)} with a unified visitor type.
 * No additional methods are defined; this type exists to satisfy
 * the required interface signature in the project specification.
 * </p>
 */
public interface TaskVisitor extends ITaskVisitor {
    // Marker interface extending ITaskVisitor
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\persistence\DAOProvider.java

DAOProvider

```java
package taskmanagement.persistence;

import taskmanagement.persistence.derby.EmbeddedDerbyTasksDAO;

/**
 * Provides centralized access to the application's {@link ITasksDAO}.
 * <p>
 * This class ensures that only a single {@link ITasksDAO} instance is used
 * throughout the application (Singleton pattern).
 * </p>
 */
public final class DAOProvider {

    /**
     * Private constructor to prevent instantiation of this utility class.
     */
    private DAOProvider() {
    }

    /**
     * Returns the global singleton instance of the {@link ITasksDAO}.
     * <p>
     * The returned DAO is backed by an embedded Derby implementation.
     * </p>
     *
     * @return the singleton {@link ITasksDAO} instance
     */
    public static ITasksDAO get() {
        return EmbeddedDerbyTasksDAO.getInstance();
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\persistence\derby\DerbyBootstrap.java

DerbyBootstrap

```java
package taskmanagement.persistence.derby;

import java.sql.Connection;
import java.sql.DatabaseMetaData;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

/**
 * Utility class responsible for booting the embedded Derby database,
 * creating it if necessary, and ensuring that the required schema exists.
 * <p>
 * This class is final and cannot be instantiated.
 */
public final class DerbyBootstrap {

    /**
     * Private constructor to prevent instantiation.
     */
    private DerbyBootstrap() {
    }

    /**
     * Boots the embedded Derby database, creates it if it does not already exist,
     * ensures that the required schema is present, and returns an open
     * {@link Connection}.
     *
     * @return an open {@link Connection} to the embedded Derby database
     * @throws IllegalStateException if the Derby driver cannot be loaded
     *                               or the database cannot be booted
     */
    public static Connection bootAndEnsureSchema() {
        try {
            Class.forName(DerbyConfig.DRIVER_CLASS);
        } catch (ClassNotFoundException e) {
            throw new IllegalStateException("Derby driver not found: " + DerbyConfig.DRIVER_CLASS, e);
        }

        try (Connection ignored = DriverManager.getConnection(DerbyConfig.urlCreate())) {
            // Attempt to create the database if it does not exist
        } catch (SQLException createEx) {
            // Ignore exceptions if database already exists
        }

        try {
            Connection conn = DriverManager.getConnection(DerbyConfig.urlBoot());
            ensureSchema(conn); // Ensure required tables are present
            return conn;
        } catch (SQLException bootEx) {
            throw new IllegalStateException("Failed to boot Derby DB", bootEx);
        }
    }

    /**
     * Ensures the required tables exist in the database.
     * Creates missing tables if they are not present.
     *
     * @param conn an open {@link Connection} to the database
```

DerbyBootstrap

```java
     * @throws SQLException if a database access error occurs
     */
    private static void ensureSchema(Connection conn) throws SQLException {
        if (!tableExists(conn, DerbyConfig.TABLE_TASKS)) {
            try (Statement st = conn.createStatement()) {
                st.executeUpdate(
                        "CREATE TABLE " + DerbyConfig.TABLE_TASKS + " (" +
                                "  id INT PRIMARY KEY," +
                                "  title VARCHAR(255) NOT NULL," +
                                "  description VARCHAR(2000) NOT NULL," +
                                "  state VARCHAR(32) NOT NULL" +
                                ")"
                );
            }
        }
    }

    /**
     * Checks whether a table with the specified name exists in the database.
     *
     * @param conn  an open {@link Connection} to the database
     * @param table the table name to check
     * @return {@code true} if the table exists, {@code false} otherwise
     * @throws SQLException if a database access error occurs
     */
    private static boolean tableExists(Connection conn, String table) throws SQLException {
        DatabaseMetaData meta = conn.getMetaData();
        try (ResultSet rs = meta.getTables(null, null, table.toUpperCase(), null)) {
            return rs.next();
        }
    }

    /**
     * Attempts to shut down the embedded Derby database and closes
     * the given {@link Connection}. Any shutdown-related exceptions
     * are ignored as Derby throws an exception when shutdown succeeds.
     *
     * @param conn an open {@link Connection} to close, may be {@code null}
     */
    public static void shutdownQuietly(Connection conn) {
        if (conn != null) {
            try {
                conn.close();
            } catch (SQLException ignored) {
                // ignore close failures
            }
        }
        try {
            DriverManager.getConnection(DerbyConfig.urlShutdown());
        } catch (SQLException ignored) {
            // Derby throws an exception on successful shutdown
        }
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\persistence\derby\DerbyConfig.java

DerbyConfig

```java
package taskmanagement.persistence.derby;

/**
 * Provides constants and utility methods for configuring
 * the embedded Apache Derby database used by the application.
 * <p>
 * This class centralizes the database name, driver class,
 * and connection URL construction for create, boot, and shutdown phases.
 * It cannot be instantiated.
 */
public final class DerbyConfig {

    /**
     * The relative name of the embedded Derby database.
     */
    public static final String DB_NAME = "tasksdb";

    /**
     * The fully qualified class name of the embedded Derby JDBC driver.
     */
    public static final String DRIVER_CLASS = "org.apache.derby.jdbc.EmbeddedDriver";

    /**
     * The name of the tasks table in the database schema.
     */
    public static final String TABLE_TASKS = "tasks";

    /**
     * Private constructor to prevent instantiation.
     */
    private DerbyConfig() {
    }

    /**
     * Builds the JDBC URL for creating the database if it does not already exist.
     *
     * @return a JDBC URL string for creating the Derby database
     */
    public static String urlCreate() {
        return "jdbc:derby:" + DB_NAME + ";create=true";
    }

    /**
     * Builds the JDBC URL for booting an existing Derby database.
     *
     * @return a JDBC URL string for booting the Derby database
     */
    public static String urlBoot() {
        return "jdbc:derby:" + DB_NAME;
    }

    /**
     * Builds the JDBC URL for shutting down the entire embedded Derby system.
     * Note: a successful shutdown typically throws a {@link java.sql.SQLException}.
     *
     * @return a JDBC URL string for shutting down Derby
     */
    public static String urlShutdown() {
        return "jdbc:derby:;shutdown=true";
```

DerbyConfig

```
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\persistence\derby\EmbeddedDerbyTasksDAO.java

EmbeddedDerbyTasksDAO

```java
package taskmanagement.persistence.derby;

import taskmanagement.domain.ITask;
import taskmanagement.domain.Task;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.exceptions.ValidationException;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;
import java.util.Objects;

/**
 * Data Access Object (DAO) implementation backed by embedded Apache Derby.
 * <p>
 * This implementation maintains a single connection created by {@link DerbyBootstrap}
 * and follows the Singleton pattern via {@link #getInstance()}.
 * It performs only persistence logic and adheres to the {@link ITasksDAO} contract.
 */
public final class EmbeddedDerbyTasksDAO implements ITasksDAO {

    private static EmbeddedDerbyTasksDAO instance;

    /**
     * Returns the singleton instance of this DAO.
     *
     * @return the singleton {@code EmbeddedDerbyTasksDAO} instance
     */
    public static synchronized EmbeddedDerbyTasksDAO getInstance() {
        if (instance == null) {
            instance = new EmbeddedDerbyTasksDAO();
        }
        return instance;
    }

    private final Connection conn;

    EmbeddedDerbyTasksDAO() {
        this.conn = DerbyBootstrap.bootAndEnsureSchema();
    }

    /**
     * Retrieves all tasks ordered by ID.
     *
     * @return an array containing all tasks (never {@code null})
     * @throws TasksDAOException if a database or mapping error occurs
     */
    @Override
    public ITask[] getTasks() throws TasksDAOException {
        final String sql = "SELECT id, title, description, state FROM TASKS ORDER BY id";
        final List<ITask> list = new ArrayList<>();
        try (PreparedStatement ps = conn.prepareStatement(sql);
             ResultSet rs = ps.executeQuery()) {
```

```java
            while (rs.next()) {
                list.add(mapRow(rs));
            }
            return list.toArray(new ITask[0]);
        } catch (SQLException | ValidationException e) {
            throw new TasksDAOException("Failed to fetch tasks", e);
        }
    }

    /**
     * Retrieves a single task by its ID.
     *
     * @param id the task ID
     * @return the task matching the given ID
     * @throws TasksDAOException if the task is not found or an error occurs
     */
    @Override
    public ITask getTask(int id) throws TasksDAOException {
        final String sql = "SELECT id, title, description, state FROM TASKS WHERE id=?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            try (ResultSet rs = ps.executeQuery()) {
                if (!rs.next()) {
                    throw new TasksDAOException("Task not found: id=" + id);
                }
                return mapRow(rs);
            }
        } catch (SQLException | ValidationException e) {
            throw new TasksDAOException("getTask failed for id=" + id, e);
        }
    }

    /**
     * Inserts a new task. If {@code task.getId() <= 0}, a new ID is allocated and
     * written back into the task object when possible.
     *
     * @param task the task to insert (must not be {@code null})
     * @throws TasksDAOException if a database error or duplicate key occurs
     */
    @Override
    public void addTask(ITask task) throws TasksDAOException {
        Objects.requireNonNull(task, "task");

        int idToInsert = task.getId();
        try {
            if (idToInsert <= 0) {
                idToInsert = nextId();
            }
        } catch (SQLException e) {
            throw new TasksDAOException("Failed to allocate next id", e);
        }

        final String sql = "INSERT INTO TASKS (id, title, description, state) VALUES (?, ?, ?, ?)";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, idToInsert);
            ps.setString(2, task.getTitle());
            ps.setString(3, task.getDescription());
            ps.setString(4, task.getState().name());
            ps.executeUpdate();
```

EmbeddedDerbyTasksDAO

```
            if (task instanceof Task t) {
                if (t.getId() != idToInsert) {
                    t.setId(idToInsert);
                }
            } else if (task.getId() <= 0) {
                throw new TasksDAOException(
                        "Unsupported task implementation; cannot assign generated id to " + task.getClass().getName());
            }

        } catch (SQLException e) {
            if ("23505".equals(e.getSQLState())) { // Derby duplicate key SQLState
                throw new TasksDAOException("Task id already exists: id=" + idToInsert, e);
            }
            throw new TasksDAOException("addTask failed for id=" + idToInsert, e);
        }
    }

    /**
     * Updates an existing task by its ID.
     *
     * @param task the task containing updated values
     * @throws TasksDAOException if the task is not found or a database error occurs
     */
    @Override
    public void updateTask(ITask task) throws TasksDAOException {
        final String sql = "UPDATE TASKS SET title=?, description=?, state=? WHERE id=?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setString(1, task.getTitle());
            ps.setString(2, task.getDescription());
            ps.setString(3, task.getState().name());
            ps.setInt(4, task.getId());
            final int updated = ps.executeUpdate();
            if (updated == 0) {
                throw new TasksDAOException("Cannot update, task not found: id=" + task.getId());
            }
        } catch (SQLException e) {
            throw new TasksDAOException("updateTask failed for id=" + task.getId(), e);
        }
    }

    /**
     * Deletes all tasks.
     *
     * @throws TasksDAOException if a database error occurs
     */
    @Override
    public void deleteTasks() throws TasksDAOException {
        final String sql = "DELETE FROM TASKS";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.executeUpdate();
        } catch (SQLException e) {
            throw new TasksDAOException("Failed to delete all tasks", e);
        }
    }

    /**
     * Deletes a single task by its ID.
     *
```

```java
     * @param id the task ID
     * @throws TasksDAOException if the task is not found or a database error occurs
     */
    @Override
    public void deleteTask(int id) throws TasksDAOException {
        final String sql = "DELETE FROM TASKS WHERE id=?";
        try (PreparedStatement ps = conn.prepareStatement(sql)) {
            ps.setInt(1, id);
            final int deleted = ps.executeUpdate();
            if (deleted == 0) {
                throw new TasksDAOException("Cannot delete, task not found: id=" + id);
            }
        } catch (SQLException e) {
            throw new TasksDAOException("deleteTask failed for id=" + id, e);
        }
    }

    private Task mapRow(ResultSet rs) throws SQLException, ValidationException {
        final int id = rs.getInt("id");
        final String title = rs.getString("title");
        final String description = rs.getString("description");
        final String stateStr = rs.getString("state");
        final TaskState state = TaskState.valueOf(stateStr);
        return new Task(id, title, description, state);
    }

    private int nextId() throws SQLException {
        final String sql = "SELECT COALESCE(MAX(id), 0) + 1 FROM TASKS";
        try (Statement st = conn.createStatement();
             ResultSet rs = st.executeQuery(sql)) {
            rs.next();
            return rs.getInt(1);
        }
    }

    /**
     * Shuts down the embedded Derby database quietly and closes the underlying connection.
     */
    public void shutdown() {
        DerbyBootstrap.shutdownQuietly(conn);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\persistence\ITasksDAO.java

ITasksDAO

```java
package taskmanagement.persistence;

import taskmanagement.domain.ITask;

/**
 * Data Access Object (DAO) contract for task persistence.
 * <p>
 * The API follows the project requirements (array-based results).
 * Implementations must not return {@code null} for a missing entity; they
 * should throw {@link TasksDAOException} instead.
 * </p>
 */
public interface ITasksDAO {

    /**
     * Retrieves all tasks currently stored.
     *
     * @return a non-{@code null} array of tasks (may be empty)
     * @throws TasksDAOException if a persistence error occurs
     */
    ITask[] getTasks() throws TasksDAOException;

    /**
     * Retrieves a task by its unique identifier.
     *
     * @param id the task identifier
     * @return the matching task (never {@code null})
     * @throws TasksDAOException if the task is not found or a persistence error occurs
     */
    ITask getTask(int id) throws TasksDAOException;

    /**
     * Persists a new task.
     *
     * @param task the task to add
     * @throws TasksDAOException if a persistence error occurs
     */
    void addTask(ITask task) throws TasksDAOException;

    /**
     * Updates an existing task (matched by its identifier).
     *
     * @param task the task containing updated fields
     * @throws TasksDAOException if the task is not found or a persistence error occurs
     */
    void updateTask(ITask task) throws TasksDAOException;

    /**
     * Deletes all tasks from the underlying storage.
     *
     * @throws TasksDAOException if the operation fails
     */
    void deleteTasks() throws TasksDAOException;

    /**
     * Deletes a single task by its unique identifier.
     *
     * @param id the task identifier
     * @throws TasksDAOException if the task is not found or a persistence error occurs
```

ITasksDAO

```java
     */
    void deleteTask(int id) throws TasksDAOException;
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\persistence\TasksDAOException.java

TasksDAOException

```java
package taskmanagement.persistence;

/**
 * Exception type for persistence and DAO-related errors specific to
 * the Tasks Management Application.
 * <p>
 * This custom exception ensures that persistence failures are clearly
 * distinguished from generic runtime exceptions.
 * </p>
 */
public class TasksDAOException extends RuntimeException {

    /**
     * Creates a new {@code TasksDAOException} with a descriptive message.
     *
     * @param message the detail message
     */
    public TasksDAOException(String message) {
        super(message);
    }

    /**
     * Creates a new {@code TasksDAOException} with a descriptive message
     * and a root cause.
     *
     * @param message the detail message
     * @param cause   the underlying cause of the exception
     */
    public TasksDAOException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

RowsPropertyAdapter

```java
package taskmanagement.ui.adapters;

import taskmanagement.application.viewmodel.TasksViewModel.RowDTO;
import taskmanagement.application.viewmodel.events.Property;
import taskmanagement.domain.ITask;

import java.util.*;

/**
 * Adapter that converts a {@code Property<List<RowDTO>>} from the ViewModel
 * into a {@code Property<List<ITask>>} suitable for UI widgets.
 * <p>
 * The adapter emits a new immutable list instance on every ViewModel update to
 * ensure UI listeners are notified, and maintains per-id proxy objects to
 * preserve selection and focus across refreshes.
 */
public final class RowsPropertyAdapter {

    private final Property<List<RowDTO>> vmRows;
    private final Property<List<ITask>> uiRows;

    /** Cache proxies by task id to keep selection stable after refresh. */
    private final Map<Integer, UiTaskProxy> cache = new HashMap<>();

    /**
     * Creates the adapter and starts listening to ViewModel changes.
     *
     * @param vmRows the ViewModel property that exposes row snapshots; must not be {@code null}
     * @throws NullPointerException if {@code vmRows} is {@code null}
     */
    public RowsPropertyAdapter(final Property<List<RowDTO>> vmRows) {
        this.vmRows = Objects.requireNonNull(vmRows, "vmRows");
        this.uiRows = new Property<>(List.of());
        rebuild(this.vmRows.getValue());
        this.vmRows.addListener((oldValue, newValue) -> rebuild(newValue));
    }

    /**
     * Returns the UI-facing property for binding in views.
     *
     * @return an observable property of an immutable {@code List<ITask>}
     */
    public Property<List<ITask>> asProperty() {
        return uiRows;
    }

    /**
     * Returns the current adapted UI list.
     *
     * @return the current UI list; never {@code null}
     */
    public List<ITask> getCurrentUiRows() {
        final List<ITask> v = uiRows.getValue();
        return v != null ? v : List.of();
    }

    /**
     * Rebuilds the UI list from the latest ViewModel rows and publishes a new list instance.
     *
```

RowsPropertyAdapter

```java
     * @param rows latest rows from the ViewModel; may be {@code null} or empty
     */
    private void rebuild(final List<RowDTO> rows) {
        if (rows == null || rows.isEmpty()) {
            cache.clear();
            uiRows.setValue(List.of());
            return;
        }

        final List<ITask> fresh = new ArrayList<>(rows.size());
        final Set<Integer> present = new HashSet<>(rows.size());

        for (RowDTO r : rows) {
            final int id = r.id();
            present.add(id);

            UiTaskProxy p = cache.get(id);
            if (p == null) {
                p = new UiTaskProxy(r);
                cache.put(id, p);
            } else {
                p.updateFrom(r); // keep existing proxy so selection/focus survive edits
            }
            fresh.add(p);
        }

        cache.keySet().removeIf(id -> !present.contains(id));
        uiRows.setValue(List.copyOf(fresh));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\adapters\TasksViewApiAdapter.java

TasksViewApiAdapter

```java
package taskmanagement.ui.adapters;

import taskmanagement.application.viewmodel.ExportFormat;
import taskmanagement.application.viewmodel.TasksViewModel;
import taskmanagement.application.viewmodel.events.Property;
import taskmanagement.application.viewmodel.sort.SortStrategy;
import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.filter.ITaskFilter;
import taskmanagement.domain.visitor.reports.ByStateCount;
import taskmanagement.ui.api.TasksViewAPI;

import java.nio.file.Path;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

/**
 * Adapter that bridges the UI-facing {@link TasksViewAPI} to the {@link TasksViewModel}.
 * <p>
 * Exposes observable properties and delegates UI commands to the ViewModel while
 * preserving UI-friendly types. Centralizes exception handling so UI code remains simple.
 */
public final class TasksViewApiAdapter implements TasksViewAPI {

    private final TasksViewModel vm;
    private final RowsPropertyAdapter rowsAdapter;
    private final RowsPropertyAdapter filteredRowsAdapter;

    /**
     * Creates a new adapter around the given ViewModel.
     *
     * @param vm the ViewModel instance; must not be {@code null}
     * @throws NullPointerException if {@code vm} is {@code null}
     */
    public TasksViewApiAdapter(TasksViewModel vm) {
        this.vm = Objects.requireNonNull(vm, "vm");
        this.rowsAdapter = new RowsPropertyAdapter(vm.rowsProperty());
        this.filteredRowsAdapter = new RowsPropertyAdapter(vm.filteredRowsProperty());
    }

    /**
     * Returns an observable list of UI-facing tasks.
     *
     * @return the property containing {@code List<ITask>}
     */
    @Override
    public Property<List<ITask>> tasksProperty() {
        return rowsAdapter.asProperty();
    }

    /**
     * Returns an observable list of UI-facing tasks after filtering.
     *
     * @return the property containing filtered {@code List<ITask>}
     */
    @Override
    public Property<List<ITask>> filteredTasksProperty() {
        return filteredRowsAdapter.asProperty();
```

TasksViewApiAdapter

```java
    }

    /**
     * Indicates whether an undo operation is currently available.
     *
     * @return property reflecting undo availability
     */
    @Override
    public Property<Boolean> canUndoProperty() {
        return vm.canUndoProperty();
    }

    /**
     * Indicates whether a redo operation is currently available.
     *
     * @return property reflecting redo availability
     */
    @Override
    public Property<Boolean> canRedoProperty() {
        return vm.canRedoProperty();
    }

    /**
     * Reloads tasks from the underlying data source.
     *
     * @return {@code null} (for fluent API compatibility)
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void reload() {
        return safeVm(() -> { vm.reload(); return null; }, "reload");
    }

    /**
     * Deletes all tasks.
     *
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void deleteAll() {
        return safeVm(() -> { vm.deleteAll(); return null; }, "deleteAll");
    }

    /**
     * Deletes the tasks with the provided identifiers.
     *
     * @param ids task identifiers
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void deleteTasks(int... ids) {
        return safeVm(() -> { vm.deleteTasks(ids); return null; }, "deleteTasks");
    }

    /**
     * Advances the lifecycle state of a task by its identifier.
     *
```

TasksViewApiAdapter

```java
     * @param id task identifier
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void advanceState(int id) {
        return safeVm(() -> { vm.advanceState(id); return null; }, "advanceState");
    }

    /**
     * Marks a task with a specific state.
     *
     * @param id    task identifier
     * @param state new state to set
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void markState(int id, TaskState state) {
        return safeVm(() -> { vm.markState(id, state); return null; }, "markState");
    }

    /**
     * Adds a new task using values from a UI-level {@link ITask}.
     *
     * @param task UI task whose title, description, and state are used
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void addTask(ITask task) {
        return safeVm(() -> {
            vm.addTask(task.getTitle(), task.getDescription(), task.getState());
            return null;
        }, "addTask");
    }

    /**
     * Updates an existing task using values from a UI-level {@link ITask}.
     *
     * @param task UI task providing id, title, description, and state
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void updateTask(ITask task) {
        return safeVm(() -> {
            vm.updateTask(task.getId(), task.getTitle(), task.getDescription(), task.getState());
            vm.reload();
            return null;
        }, "updateTask");
    }

    /**
     * Performs an undo operation if available.
     *
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
```

TasksViewApiAdapter

```java
    @Override
    public Void undo() {
        return safeVm(() -> { vm.undo(); return null; }, "undo");
    }

    /**
     * Performs a redo operation if available.
     *
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void redo() {
        return safeVm(() -> { vm.redo(); return null; }, "redo");
    }

    /**
     * Applies a task filter.
     *
     * @param filter the filter to apply
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void setFilter(ITaskFilter filter) {
        return safeVm(() -> { vm.setFilter(filter); return null; }, "setFilter");
    }

    /**
     * Clears the active task filter.
     *
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void clearFilter() {
        return safeVm(() -> { vm.clearFilter(); return null; }, "clearFilter");
    }

    /**
     * Sets the sorting strategy for tasks.
     *
     * @param strategy the sorting strategy to use
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void setSortStrategy(SortStrategy strategy) {
        return safeVm(() -> { vm.setSortStrategy(strategy); return null; }, "setSortStrategy");
    }

    /**
     * Computes counts of tasks by state.
     *
     * @param useFiltered whether to use the filtered list
     * @return a {@link ByStateCount} report
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\adapters\TasksViewApiAdapter.java

TasksViewApiAdapter

```java
    public ByStateCount getCountsByState(boolean useFiltered) {
        return safeVm(() -> vm.getCountsByState(useFiltered), "getCountsByState");
    }

    /**
     * Exports tasks to a file in the selected format.
     *
     * @param path        output file path
     * @param format      export format
     * @param useFiltered whether to use the filtered list
     * @param ids         optional subset of task ids to export
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void exportTasks(Path path, ExportFormat format, boolean useFiltered, List<Integer> ids) {
        return safeVm(() -> { vm.exportTasks(path, format, useFiltered, ids); return null; }, "exportTasks");
    }

    /**
     * Exports a by-state report to a file in the selected format.
     *
     * @param path        output file path
     * @param format      export format
     * @param useFiltered whether to use the filtered list
     * @param ids         optional subset of task ids to include
     * @return {@code null}
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Void exportByStateReport(Path path, ExportFormat format, boolean useFiltered, List<Integer> ids) {
        return safeVm(() -> { vm.exportByStateReport(path, format, useFiltered, ids); return null; }, "exportByStateReport");
    }

    /**
     * Finds a view-model row by its identifier.
     *
     * @param id task identifier
     * @return an {@link Optional} containing the matching row, if found
     * @throws IllegalStateException if the ViewModel operation fails
     */
    @Override
    public Optional<TasksViewModel.RowDTO> findRowById(int id) {
        return safeVm(() -> vm.findRowById(id), "findRowById");
    }

    /**
     * Executes a ViewModel operation and wraps any thrown exception in an {@link IllegalStateException}.
     *
     * @param action operation to execute
     * @param opName operation name for diagnostics
     * @param <T>    return type
     * @return the operation result
     * @throws IllegalStateException if {@code action} throws an exception
     */
    private static <T> T safeVm(CheckedSupplier<T> action, String opName) {
        try {
            return action.get();
        } catch (Exception e) {
```

TasksViewApiAdapter

```java
            throw new IllegalStateException("ViewModel operation failed: " + opName, e);
        }
    }

    @FunctionalInterface
    private interface CheckedSupplier<T> {
        T get() throws Exception;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\adapters\UiTaskProxy.java

UiTaskProxy

```java
package taskmanagement.ui.adapters;

import taskmanagement.application.viewmodel.TasksViewModel.RowDTO;
import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.visitor.TaskVisitor;

import java.util.Objects;

/**
 * UI-side proxy that implements {@link ITask} and mirrors a single
 * {@link RowDTO} published by the {@code TasksViewModel}.
 * <p>
 * Instances are intended to be selection-stable in the UI: reuse the same proxy
 * per task id and call {@link #updateFrom(RowDTO)} when new snapshots arrive.
 * This class contains no DAO or domain logic.
 */
public final class UiTaskProxy implements ITask {

    private int id;
    private String title;
    private String description;
    private TaskState state;

    /**
     * Constructs a proxy from a ViewModel row snapshot.
     *
     * @param row non-null row DTO from the ViewModel
     * @throws NullPointerException if {@code row} is {@code null}
     */
    public UiTaskProxy(final RowDTO row) {
        updateFrom(row);
    }

    /**
     * Updates this proxy with values from a ViewModel row snapshot.
     * All fields are overwritten.
     *
     * @param row non-null row DTO from the ViewModel
     * @throws NullPointerException if {@code row} is {@code null}
     */
    public void updateFrom(final RowDTO row) {
        Objects.requireNonNull(row, "row");
        this.id = row.id();
        this.title = row.title();
        this.description = row.description();

        // Defensive mapping from row.state() to TaskState with a safe fallback.
        TaskState mapped;
        try {
            final String s = String.valueOf(row.state());
            mapped = TaskState.valueOf(s.trim());
        } catch (Exception ex) {
            mapped = TaskState.ToDo;
        }
        this.state = mapped;
    }

    /**
```

UiTaskProxy

```java
     * {@inheritDoc}
     */
    @Override
    public int getId() {
        return id;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String getTitle() {
        return title;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public String getDescription() {
        return description;
    }

    /**
     * {@inheritDoc}
     */
    @Override
    public TaskState getState() {
        return state;
    }

    /**
     * No-op for UI proxies; domain visitors are not applied on the UI layer.
     *
     * @param v the visitor instance (ignored)
     */
    @Override
    public void accept(final TaskVisitor v) { /* intentionally no-op */ }

    /**
     * Returns a concise textual representation for debugging.
     *
     * @return a string containing id, title, and state
     */
    @Override
    public String toString() {
        return "UiTaskProxy{id=" + id + ", title='" + title + "', state=" + state + "}";
    }
}
```

TasksViewAPI

```java
package taskmanagement.ui.api;

import taskmanagement.application.viewmodel.TasksViewModel;
import taskmanagement.application.viewmodel.sort.SortStrategy;
import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.filter.ITaskFilter;
import taskmanagement.domain.visitor.reports.ByStateCount;
import taskmanagement.application.viewmodel.ExportFormat;
import taskmanagement.application.viewmodel.events.Property;

import java.nio.file.Path;
import java.util.List;
import java.util.Optional;

/**
 * UI-facing abstraction over the {@link TasksViewModel}.
 * <p>
 * The Swing layer depends only on this interface and never touches the DAO or
 * domain model directly. Methods return {@code Void} (or DTOs) to keep the view
 * free from checked exceptions.
 */
public interface TasksViewAPI {

    // ---------------------------------------------------------------------
    // Properties
    // ---------------------------------------------------------------------

    /**
     * Returns an observable property of all tasks (unfiltered), suitable for UI binding.
     *
     * @return property containing {@code List<ITask>} of all tasks
     */
    Property<List<ITask>> tasksProperty();

    /**
     * Returns an observable property of tasks after applying the current filter.
     *
     * @return property containing filtered {@code List<ITask>}
     */
    Property<List<ITask>> filteredTasksProperty();

    /**
     * Returns an observable flag indicating whether an undo operation is possible.
     *
     * @return property reflecting undo availability
     */
    Property<Boolean> canUndoProperty();

    /**
     * Returns an observable flag indicating whether a redo operation is possible.
     *
     * @return property reflecting redo availability
     */
    Property<Boolean> canRedoProperty();

    // ---------------------------------------------------------------------
    // Core Operations
    // ---------------------------------------------------------------------
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\api\TasksViewAPI.java

TasksViewAPI

```
    /**
     * Reloads tasks from the persistence layer into the ViewModel.
     *
     * @return {@code null}
     */
    Void reload();

    /**
     * Deletes all tasks.
     *
     * @return {@code null}
     */
    Void deleteAll();

    /**
     * Deletes a set of tasks by their identifiers.
     *
     * @param ids task identifiers to delete
     * @return {@code null}
     */
    Void deleteTasks(int... ids);

    /**
     * Advances a task to its next legal {@link TaskState}.
     *
     * @param id task identifier
     * @return {@code null}
     */
    Void advanceState(int id);

    /**
     * Marks a task with an explicit target {@link TaskState}.
     *
     * @param id    task identifier
     * @param state target state
     * @return {@code null}
     */
    Void markState(int id, TaskState state);

    /**
     * Adds a new task using a UI-level {@link ITask} proxy. Implementations extract
     * title, description, and state as needed.
     *
     * @param task UI proxy carrying task data
     * @return {@code null}
     */
    Void addTask(ITask task);

    /**
     * Updates an existing task using a UI-level {@link ITask} proxy. Implementations
     * use {@code task.getId()} as the key.
     *
     * @param task UI proxy carrying updated task data
     * @return {@code null}
     */
    Void updateTask(ITask task);

    /**
```

```
     * Performs an undo operation if available.
     *
     * @return {@code null}
     */
    Void undo();

    /**
     * Performs a redo operation if available.
     *
     * @return {@code null}
     */
    Void redo();

    // ---------------------------------------------------------------------
    // Filtering / Sorting
    // ---------------------------------------------------------------------

    /**
     * Applies a composed filter (AND/OR combinator) to the tasks view. Implementations
     * should update {@link #filteredTasksProperty()} accordingly.
     *
     * @param filter combinator filter to apply
     * @return {@code null}
     */
    Void setFilter(ITaskFilter filter);

    /**
     * Clears the active filter.
     *
     * @return {@code null}
     */
    Void clearFilter();

    /**
     * Sets (or clears with {@code null}) the sorting strategy used for presentation.
     *
     * @param strategy sorting strategy, or {@code null} to clear
     * @return {@code null}
     */
    Void setSortStrategy(SortStrategy strategy);

    // ---------------------------------------------------------------------
    // Reporting
    // ---------------------------------------------------------------------

    /**
     * Computes counts of tasks by state, optionally on the filtered subset.
     *
     * @param useFiltered {@code true} to compute on filtered tasks; {@code false} to use all tasks
     * @return a {@link ByStateCount} report DTO
     */
    ByStateCount getCountsByState(boolean useFiltered);

    /**
     * Exports tasks to the given path in the requested format.
     *
     * @param path        target file path
     * @param format      export format (e.g., CSV or TXT)
     * @param useFiltered whether to export the filtered subset
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\api\TasksViewAPI.java

TasksViewAPI

```java
     * @param ids          optional explicit list of ids to export; may be {@code null} or empty
     * @return {@code null}
     */
    Void exportTasks(Path path, ExportFormat format, boolean useFiltered, List<Integer> ids);

    /**
     * Exports a "count by state" report.
     *
     * @param path        target file path
     * @param format      export format (e.g., CSV or TXT)
     * @param useFiltered whether to export the filtered subset
     * @param ids         optional explicit list of ids to include; may be {@code null} or empty
     * @return {@code null}
     */
    Void exportByStateReport(Path path, ExportFormat format, boolean useFiltered, List<Integer> ids);

    // ---------------------------------------------------------------------
    // Lookup for dialogs
    // ---------------------------------------------------------------------

    /**
     * Finds a task row by its identifier for dialog prefill.
     *
     * @param id task identifier
     * @return optional {@link TasksViewModel.RowDTO} if found
     */
    Optional<TasksViewModel.RowDTO> findRowById(int id);
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\chrome\WindowChrome.java

WindowChrome

```java
package taskmanagement.ui.chrome;

import taskmanagement.ui.styles.AppTheme;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.geom.RoundRectangle2D;

/**
 * Utility methods for borderless windows with rounded corners and drag handling.
 * <p>
 * Provides helpers to make a frame undecorated with rounded corners and to
 * install a drag handler so a component can move the window.
 */
public final class WindowChrome {

    private WindowChrome() {}

    /**
     * Makes the frame borderless and applies rounded corners.
     * <p>
     * Sets the frame to undecorated, applies a transparent background, and
     * keeps a rounded shape synchronized with size changes using
     * {@link AppTheme#WINDOW_CORNER_ARC}.
     *
     * @param frame the frame to modify; no action if {@code null}
     */
    public static void makeBorderlessWithRoundedCorners(JFrame frame) {
        if (frame == null) return;

        try {
            frame.setUndecorated(true);
        } catch (IllegalComponentStateException ignore) {
            // If already visible/packed as decorated, caller should re-create before showing.
        }

        frame.setBackground(new Color(0, 0, 0, 0));
        applyRoundedShape(frame);

        frame.addComponentListener(new ComponentAdapter() {
            @Override public void componentResized(ComponentEvent e) { applyRoundedShape(frame); }
            @Override public void componentShown(ComponentEvent e)   { applyRoundedShape(frame); }
        });
    }

    /**
     * Installs a drag handler so dragging the given component moves the frame.
     * Intended for use with borderless windows.
     *
     * @param frame      the frame to move; no action if {@code null}
     * @param dragHandle the component acting as a drag handle; no action if {@code null}
     */
    public static void installDragHandler(JFrame frame, JComponent dragHandle) {
        if (frame == null || dragHandle == null) return;
```

WindowChrome

```java
        final Point[] origin = new Point[1];

        MouseAdapter ma = new MouseAdapter() {
            @Override public void mousePressed(MouseEvent e) {
                origin[0] = e.getPoint();
            }
            @Override public void mouseDragged(MouseEvent e) {
                if (origin[0] != null) {
                    Point p = e.getLocationOnScreen();
                    Insets ins = frame.getInsets();
                    frame.setLocation(p.x - origin[0].x - ins.left, p.y - origin[0].y - ins.top);
                }
            }
            @Override public void mouseReleased(MouseEvent e) {
                origin[0] = null;
            }
        };

        dragHandle.addMouseListener(ma);
        dragHandle.addMouseMotionListener(ma);
    }

    /** Applies a rounded rectangle shape using {@link AppTheme#WINDOW_CORNER_ARC}. */
    private static void applyRoundedShape(JFrame frame) {
        int arc = AppTheme.WINDOW_CORNER_ARC;
        int w = frame.getWidth();
        int h = frame.getHeight();
        if (w <= 0 || h <= 0) return;

        Shape round = new RoundRectangle2D.Double(0, 0, w, h, arc, arc);
        try {
            frame.setShape(round);
        } catch (UnsupportedOperationException ignored) {
            // On some platforms/VMs shapes are unsupported; ignore gracefully.
        }
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\AboutDialog.java

AboutDialog

```java
package taskmanagement.ui.dialogs;

import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.util.UiUtils;

import javax.swing.*;
import java.awt.*;
import java.awt.datatransfer.StringSelection;
import java.awt.event.ActionEvent;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.net.URI;

/**
 * Modal “About” dialog for the application.
 * <p>
 * Presentation-only (MVVM-safe): contains no DAO or model access.
 */
public final class AboutDialog extends JDialog {

    private JButton okButton;

    /**
     * Creates a modal About dialog.
     *
     * @param owner the owner window; may be {@code null}
     */
    public AboutDialog(Window owner) {
        super(owner, "About", ModalityType.APPLICATION_MODAL);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setContentPane(buildContent());
        pack();
        setResizable(false);
        setLocationRelativeTo(owner);
        getRootPane().setDefaultButton(okButton);

        var im = getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
        var am = getRootPane().getActionMap();
        im.put(KeyStroke.getKeyStroke("ESCAPE"), "close");
        am.put("close", new AbstractAction() {
            @Override public void actionPerformed(ActionEvent e) { dispose(); }
        });
    }

    private JComponent buildContent() {
        final RoundedPanel root = new RoundedPanel(AppTheme.PANEL_BG, AppTheme.TB_CORNER_RADIUS);
        root.setBorder(BorderFactory.createEmptyBorder(16, 16, 16, 16));
        root.setLayout(new BorderLayout(0, 12));

        Icon infoIcon = UiUtils.loadRasterIcon(
                "/taskmanagement/ui/resources/tasks_mng.png", 40, 40);

        JLabel title = new JLabel("Tasks Management Application");
        title.setFont(new Font("Segoe UI", Font.BOLD, 18));
        title.setForeground(AppTheme.MAIN_TEXT);

        JLabel subtitle = new JLabel("Version 1.0 · Swing · MVVM · Derby Embedded");
        subtitle.setFont(new Font("Segoe UI", Font.PLAIN, 13));
        subtitle.setForeground(new Color(0x888888));

        JPanel titles = new JPanel();
        titles.setOpaque(false);
        titles.setLayout(new BoxLayout(titles, BoxLayout.Y_AXIS));
        titles.add(title);
        titles.add(Box.createVerticalStrut(4));
        titles.add(subtitle);

        JPanel header = new JPanel(new BorderLayout(10, 0));
        header.setOpaque(false);
        if (infoIcon != null) {
            header.add(new JLabel(infoIcon), BorderLayout.WEST);
        }
        header.add(titles, BorderLayout.CENTER);

        JPanel body = new JPanel();
        body.setOpaque(false);
        body.setLayout(new BoxLayout(body, BoxLayout.Y_AXIS));

        JLabel description = new JLabel(
                "<html>" +
                        "<p>A lightweight desktop app to add, edit, and organize tasks.</p>" +
                        "<p><b>Architecture:</b><br>" +
                        "• Strict MVVM separation<br>" +
                        "• Derby embedded database<br>" +
                        "• Swing UI (responsive)</p>" +
                        "</html>"
        );
        description.setForeground(AppTheme.MAIN_TEXT);
        description.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        description.setAlignmentX(Component.LEFT_ALIGNMENT);

        JLabel madeByTitle = new JLabel("Made by:");
        madeByTitle.setForeground(AppTheme.MAIN_TEXT);
        madeByTitle.setFont(new Font("Segoe UI", Font.BOLD, 12));
        madeByTitle.setAlignmentX(Component.LEFT_ALIGNMENT);

        JPanel authorsList = buildOrderedAuthors();
        authorsList.setAlignmentX(Component.LEFT_ALIGNMENT);

        body.add(description);
        body.add(Box.createVerticalStrut(8));
        body.add(madeByTitle);
        body.add(Box.createVerticalStrut(4));
        body.add(authorsList);

        JPanel actions = new JPanel(new FlowLayout(FlowLayout.RIGHT, 8, 0));
        actions.setOpaque(false);

        okButton = new JButton("OK");
        okButton.setMnemonic('O');
        okButton.getAccessibleContext().setAccessibleName("OK");
        UiUtils.styleStableHoverButton(okButton, new Color(0x2F7BFF), Color.WHITE);
        okButton.addActionListener(e -> dispose());

        actions.add(okButton);

        root.add(header, BorderLayout.NORTH);
        root.add(body, BorderLayout.CENTER);
        root.add(actions, BorderLayout.SOUTH);

        return root;
    }

    private JPanel buildOrderedAuthors() {
        JPanel list = new JPanel();
        list.setOpaque(false);
        list.setLayout(new BoxLayout(list, BoxLayout.Y_AXIS));

        JPanel row1 = new JPanel(new FlowLayout(FlowLayout.LEFT, 6, 0));
        row1.setOpaque(false);
        JLabel a1Label = new JLabel("1. Itay Vaznan  ");
        a1Label.setForeground(AppTheme.MAIN_TEXT);
        a1Label.setFont(new Font("Segoe UI", Font.BOLD, 12));
        row1.add(a1Label);
        row1.add(buildLinksPanel());
        row1.setAlignmentX(Component.LEFT_ALIGNMENT);

        JPanel row2 = new JPanel(new FlowLayout(FlowLayout.LEFT, 6, 0));
        row2.setOpaque(false);
        JLabel a2Label = new JLabel("2. Yuval Benzaquen  ");
        a2Label.setForeground(AppTheme.MAIN_TEXT);
        a2Label.setFont(new Font("Segoe UI", Font.BOLD, 12));
        row2.add(a2Label);
        row2.setAlignmentX(Component.LEFT_ALIGNMENT);

        list.add(row1);
        list.add(Box.createVerticalStrut(4));
        list.add(row2);
        return list;
    }

    private JPanel buildLinksPanel() {
        JPanel links = new JPanel(new FlowLayout(FlowLayout.LEFT, 8, 0));
        links.setOpaque(false);

        links.add(makeIconLink(
                UiUtils.loadRasterIcon("/taskmanagement/ui/resources/linkedin.png", 24, 24),
                "LinkedIn",
                "https://www.linkedin.com/in/itayvazana/"));

        links.add(makeIconLink(
                UiUtils.loadRasterIcon("/taskmanagement/ui/resources/github.png", 24, 24),
                "GitHub",
                "https://github.com/ItayVazana1"));

        links.add(makeIconLink(
                UiUtils.loadRasterIcon("/taskmanagement/ui/resources/email.png", 24, 24),
                "Email",
                "mailto:itay.vazana.b@gmail.com"));

        return links;
    }

    private JComponent makeIconLink(Icon icon, String label, String url) {
        JLabel comp;
        if (icon != null) {
```

```java
            comp = new JLabel(icon);
            comp.setToolTipText(label);
        } else {
            comp = new JLabel("<html><u>   " + label + "   </u></html>");
            comp.setForeground(new Color(0x2F7BFF));
            comp.setFont(new Font("Segoe UI", Font.PLAIN, 12));
            comp.setToolTipText(label);
        }
        comp.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        comp.addMouseListener(new MouseAdapter() {
            @Override public void mouseClicked(MouseEvent e) { openLink(url); }
        });
        return comp;
    }

    private void openLink(String url) {
        try {
            if (!Desktop.isDesktopSupported()) {
                throw new UnsupportedOperationException("Desktop API not supported");
            }
            Desktop desktop = Desktop.getDesktop();
            URI uri = new URI(url);
            String scheme = uri.getScheme();

            if ("mailto".equalsIgnoreCase(scheme)) {
                if (desktop.isSupported(Desktop.Action.MAIL)) {
                    desktop.mail(uri);
                } else {
                    throw new UnsupportedOperationException("MAIL action not supported");
                }
            } else {
                if (desktop.isSupported(Desktop.Action.BROWSE)) {
                    desktop.browse(uri);
                } else {
                    throw new UnsupportedOperationException("BROWSE action not supported");
                }
            }
        } catch (Exception ex) {
            try {
                String value = url.startsWith("mailto:") ? url.substring("mailto:".length()) : url;
                Toolkit.getDefaultToolkit().getSystemClipboard().setContents(new StringSelection(value), null);
                JOptionPane.showMessageDialog(this,
                        "Couldn't open the default app.\nCopied to clipboard: " + value,
                        "Open Link", JOptionPane.INFORMATION_MESSAGE);
            } catch (Exception ignore) {
                // intentionally ignored
            }
        }
    }

    /**
     * Shows the modal About dialog.
     *
     * @param parent a component within the parent window; may be {@code null}
     */
    public static void showDialog(Component parent) {
        Window owner = parent instanceof Window ? (Window) parent : SwingUtilities.getWindowAncestor(parent);
        AboutDialog dlg = new AboutDialog(owner);
        dlg.setVisible(true);
```

AboutDialog

```
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\ConfirmExitDialog.java

ConfirmExitDialog

```java
package taskmanagement.ui.dialogs;

import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.util.UiUtils;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;

/**
 * Modal dialog that asks the user to confirm exiting the application.
 * <p>
 * Presentation-only and MVVM-safe: contains no model or DAO access.
 */
public final class ConfirmExitDialog extends JDialog {

    private JButton okButton;
    private boolean confirmed;

    /**
     * Creates a confirm-exit dialog as a modal child of the given owner.
     *
     * @param owner the parent window; may be {@code null}
     */
    public ConfirmExitDialog(Window owner) {
        super(owner, "Exit", ModalityType.APPLICATION_MODAL);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setContentPane(buildContent());
        pack();
        setResizable(false);
        setLocationRelativeTo(owner);
        getRootPane().setDefaultButton(okButton);

        var im = getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
        var am = getRootPane().getActionMap();
        im.put(KeyStroke.getKeyStroke("ESCAPE"), "cancel");
        am.put("cancel", new AbstractAction() { @Override public void actionPerformed(ActionEvent e) { onCancel(); } });
        im.put(KeyStroke.getKeyStroke("ENTER"), "confirm");
        am.put("confirm", new AbstractAction() { @Override public void actionPerformed(ActionEvent e) { onConfirm(); } });
    }

    private JComponent buildContent() {
        RoundedPanel root = new RoundedPanel(AppTheme.BODY_BG, AppTheme.WINDOW_CORNER_ARC);
        root.setBorder(BorderFactory.createEmptyBorder(16, 18, 16, 18));
        root.setLayout(new BorderLayout(0, 12));

        final Color accentBg = new Color(0x3A0E12);
        final Color accentFg = new Color(0xFFECEC);
        RoundedPanel header = new RoundedPanel(accentBg, AppTheme.WINDOW_CORNER_ARC);
        header.setLayout(new BorderLayout(10, 8));
        header.setOpaque(true);
        header.setBorder(BorderFactory.createEmptyBorder(10, 12, 10, 12));

        Icon warnIcon = UiUtils.loadRasterIcon("/taskmanagement/ui/resources/warning.png", 22, 22);
        if (warnIcon != null) {
            header.add(new JLabel(warnIcon), BorderLayout.WEST);
        }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\ConfirmExitDialog.java

ConfirmExitDialog

```java
        JLabel title = new JLabel("Exit application?");
        title.setFont(new Font("Segoe UI", Font.BOLD, 16));
        title.setForeground(accentFg);

        JLabel subtitle = new JLabel("Unsaved work may be lost.");
        subtitle.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        subtitle.setForeground(new Color(0xFFDADA));

        JPanel titles = new JPanel();
        titles.setOpaque(false);
        titles.setLayout(new BoxLayout(titles, BoxLayout.Y_AXIS));
        titles.add(title);
        titles.add(Box.createVerticalStrut(2));
        titles.add(subtitle);
        header.add(titles, BorderLayout.CENTER);

        JPanel center = new JPanel();
        center.setOpaque(false);
        center.setLayout(new BoxLayout(center, BoxLayout.Y_AXIS));
        JLabel note = new JLabel("This will close the application.");
        note.setForeground(AppTheme.MAIN_TEXT);
        note.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        note.setAlignmentX(Component.LEFT_ALIGNMENT);
        center.add(note);

        JPanel actions = new JPanel(new FlowLayout(FlowLayout.RIGHT, 8, 0));
        actions.setOpaque(false);

        JButton cancel = new JButton("Cancel");
        UiUtils.styleStableHoverButton(cancel, AppTheme.DARK_GREY, AppTheme.MAIN_TEXT);
        cancel.addActionListener(e -> onCancel());

        okButton = new JButton("Exit");
        UiUtils.styleStableHoverButton(okButton, AppTheme.IOS_RED, Color.WHITE);
        okButton.addActionListener(e -> onConfirm());

        actions.add(cancel);
        actions.add(okButton);

        root.add(header, BorderLayout.NORTH);
        root.add(center, BorderLayout.CENTER);
        root.add(actions, BorderLayout.SOUTH);
        return root;
    }

    private void onConfirm() {
        confirmed = true;
        dispose();
    }

    private void onCancel() {
        confirmed = false;
        dispose();
    }

    /**
     * Shows a modal confirm-exit dialog and returns the user's choice.
     *
     * @param parent any component inside the parent window; may be {@code null}
```

ConfirmExitDialog

```
     * @return {@code true} if the user confirmed exiting; {@code false} otherwise
     */
    public static boolean confirm(Component parent) {
        Window owner = (parent instanceof Window) ? (Window) parent : SwingUtilities.getWindowAncestor(parent);
        ConfirmExitDialog dlg = new ConfirmExitDialog(owner);
        dlg.setVisible(true);
        return dlg.confirmed;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\ExportDialog.java

ExportDialog

```java
package taskmanagement.ui.dialogs;

import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.util.UiUtils;
import taskmanagement.application.viewmodel.ExportFormat;

import javax.swing.*;
import javax.swing.filechooser.FileNameExtensionFilter;
import java.awt.*;
import java.io.File;
import java.util.Locale;
import java.util.Optional;

/**
 * Modal dialog for configuring task export (file destination and format).
 * <p>
 * Presentation-only (MVVM-safe). The caller is responsible for invoking the
 * ViewModel export command based on the returned {@link #showDialog(Component)} result.
 */
public final class ExportDialog extends JDialog {

    /**
     * Result of the export configuration chosen by the user.
     *
     * @param file         the target file
     * @param format       the selected export format
     * @param onlyFiltered legacy flag retained for API compatibility (always {@code false})
     */
    public static record ExportResult(File file, ExportFormat format, boolean onlyFiltered) {}

    private final JTextField pathField = new JTextField(28);
    private final JComboBox<ExportFormat> formatCombo = new JComboBox<>(ExportFormat.values());
    private JButton exportButton;
    private boolean confirmed;

    /**
     * Constructs the export dialog as a modal child of the given owner.
     *
     * @param owner parent window; may be {@code null}
     */
    public ExportDialog(Window owner) {
        super(owner, "Export Tasks", ModalityType.APPLICATION_MODAL);
        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setContentPane(buildContent());
        pack();
        setResizable(false);
        setLocationRelativeTo(owner);
        if (exportButton != null) {
            getRootPane().setDefaultButton(exportButton);
        }
        getRootPane().registerKeyboardAction(e -> {
                    confirmed = false;
                    dispose();
                },
                KeyStroke.getKeyStroke("ESCAPE"),
                JComponent.WHEN_IN_FOCUSED_WINDOW
        );
    }
```

ExportDialog

```java
    private JComponent buildContent() {
        RoundedPanel root = new RoundedPanel(AppTheme.PANEL_BG, AppTheme.WINDOW_CORNER_ARC);
        root.setLayout(new BorderLayout(0, 12));
        root.setBorder(BorderFactory.createEmptyBorder(16, 18, 16, 18));

        JLabel title = new JLabel("Export Tasks");
        title.setFont(AppTheme.CTRL_BUTTON_FONT.deriveFont(Font.BOLD, 16f));
        title.setForeground(AppTheme.MAIN_TEXT);

        JPanel form = new JPanel();
        form.setOpaque(false);
        form.setLayout(new GridBagLayout());
        GridBagConstraints gc = new GridBagConstraints();
        gc.insets = new Insets(6, 0, 6, 0);
        gc.fill = GridBagConstraints.HORIZONTAL;
        gc.weightx = 1.0;
        gc.gridx = 0;

        gc.gridy = 0;
        form.add(makeLabel("File:"), gc);
        gc.gridy = 1;
        try {
            UiUtils.styleTextFieldForDarkCentered(pathField);
        } catch (Throwable ignored) { }
        form.add(pathField, gc);

        JButton browseBtn = new JButton("Browse…");
        try {
            UiUtils.styleStableHoverButton(browseBtn, AppTheme.TB_SHOW_SELECTED_BG, AppTheme.MAIN_TEXT);
        } catch (Throwable ignored) { }
        browseBtn.addActionListener(e -> chooseFile());
        gc.gridy = 2;
        form.add(browseBtn, gc);

        gc.gridy = 3;
        form.add(makeLabel("Format:"), gc);
        gc.gridy = 4;

        formatCombo.setRenderer(new DefaultListCellRenderer() {
            @Override public Component getListCellRendererComponent(JList<?> list, Object value, int index,
                                                                    boolean isSelected, boolean cellHasFocus) {
                String text = (value instanceof ExportFormat ef)
                        ? switch (ef) { case CSV -> "CSV (Comma-Separated)"; case TXT -> "TXT (Plain Text)"; }
                        : String.valueOf(value);
                return super.getListCellRendererComponent(list, text, index, isSelected, cellHasFocus);
            }
        });
        formatCombo.addActionListener(e -> harmonizePathWithSelectedFormat());
        form.add(formatCombo, gc);

        JPanel actions = new JPanel(new FlowLayout(FlowLayout.RIGHT, 8, 0));
        actions.setOpaque(false);

        exportButton = new JButton("Export");
        JButton cancelBtn = new JButton("Cancel");

        try {
            UiUtils.styleStableHoverButton(exportButton, AppTheme.TB_FILTER_APPLY_BG, AppTheme.MAIN_TEXT);
```

ExportDialog

```java
            UiUtils.styleStableHoverButton(cancelBtn, AppTheme.TB_SORT_RESET_BG, AppTheme.MAIN_TEXT);
        } catch (Throwable ignored) { }

        exportButton.addActionListener(e -> onExport());
        cancelBtn.addActionListener(e -> {
            confirmed = false;
            dispose();
        });

        actions.add(cancelBtn);
        actions.add(exportButton);

        root.add(title, BorderLayout.NORTH);
        root.add(form, BorderLayout.CENTER);
        root.add(actions, BorderLayout.SOUTH);

        return root;
    }

    private JLabel makeLabel(String text) {
        JLabel l = new JLabel(text);
        l.setForeground(AppTheme.MAIN_TEXT);
        return l;
    }

    private void onExport() {
        String p = pathField.getText().trim();
        if (p.isEmpty()) {
            JOptionPane.showMessageDialog(
                    this,
                    "Please choose a file path.",
                    "Missing File",
                    JOptionPane.WARNING_MESSAGE
            );
            return;
        }
        ExportFormat fmt = currentFormat();
        File f = new File(p);
        f = ensureExtension(f, extFor(fmt));
        pathField.setText(f.getAbsolutePath());
        confirmed = true;
        dispose();
    }

    private void chooseFile() {
        ExportFormat fmt = currentFormat();
        String ext = extFor(fmt);

        JFileChooser chooser = new JFileChooser();
        chooser.setDialogTitle("Choose export file");
        chooser.setFileFilter(new FileNameExtensionFilter(ext.toUpperCase(Locale.ROOT) + " files", ext));

        if (pathField.getText().isBlank()) {
            chooser.setSelectedFile(new File("tasks_export." + ext));
        }

        int result = chooser.showSaveDialog(this);
        if (result == JFileChooser.APPROVE_OPTION) {
            File file = chooser.getSelectedFile();
            file = ensureExtension(file, ext);
            pathField.setText(file.getAbsolutePath());
        }
    }

    private void harmonizePathWithSelectedFormat() {
        String p = pathField.getText().trim();
        if (p.isEmpty()) return;
        String ext = extFor(currentFormat());
        File f = ensureExtension(new File(p), ext);
        pathField.setText(f.getAbsolutePath());
    }

    private ExportFormat currentFormat() {
        Object sel = formatCombo.getSelectedItem();
        return (sel instanceof ExportFormat ef) ? ef : ExportFormat.CSV;
    }

    private static String extFor(ExportFormat fmt) {
        return (fmt == ExportFormat.TXT) ? "txt" : "csv";
    }

    private static File ensureExtension(File file, String ext) {
        String name = file.getName();
        int dot = name.lastIndexOf('.');
        if (dot > 0) {
            String current = name.substring(dot + 1).toLowerCase(Locale.ROOT);
            if (!current.equals(ext)) {
                name = name.substring(0, dot) + "." + ext;
            }
        } else {
            name = name + "." + ext;
        }
        return new File(file.getParentFile() == null ? new File(".") : file.getParentFile(), name);
    }

    /**
     * Shows the export dialog and returns chosen options if confirmed.
     *
     * @param parent parent component for centering
     * @return an {@link Optional} containing {@link ExportResult} if confirmed; otherwise empty
     */
    public static Optional<ExportResult> showDialog(Component parent) {
        Window owner = (parent instanceof Window) ? (Window) parent : SwingUtilities.getWindowAncestor(parent);
        ExportDialog dlg = new ExportDialog(owner);
        dlg.setVisible(true);
        if (!dlg.confirmed) {
            return Optional.empty();
        }
        File file = new File(dlg.pathField.getText().trim());
        ExportFormat format = dlg.currentFormat();
        boolean onlyFiltered = false;
        return Optional.of(new ExportResult(file, format, onlyFiltered));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\TaskDetailsDialog.java

TaskDetailsDialog

```java
package taskmanagement.ui.dialogs;

import taskmanagement.domain.ITask;
import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.util.UiUtils;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.ActionEvent;

/**
 * Modal dialog that presents task details in a read-only, presentation-only view.
 * <p>
 * MVVM-safe: performs no DAO/model access and no state mutations.
 */
public final class TaskDetailsDialog extends JDialog {

    private JButton closeButton;

    /**
     * Shows a modal Task Details dialog for the given task.
     *
     * @param parent a component inside the owner window; may be {@code null}
     * @param task   the task to display; must not be {@code null}
     * @throws NullPointerException if {@code task} is {@code null}
     */
    public static void showDialog(Component parent, ITask task) {
        Window owner = parent instanceof Window ? (Window) parent : SwingUtilities.getWindowAncestor(parent);
        TaskDetailsDialog dlg = new TaskDetailsDialog(owner, task);
        dlg.setVisible(true);
    }

    /**
     * Creates the dialog. Prefer using {@link #showDialog(Component, ITask)}.
     *
     * @param owner window owner; may be {@code null}
     * @param task  task to display; must not be {@code null}
     * @throws NullPointerException if {@code task} is {@code null}
     */
    public TaskDetailsDialog(Window owner, ITask task) {
        super(owner, "Task Details", ModalityType.APPLICATION_MODAL);
        if (task == null) throw new NullPointerException("task");

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setContentPane(buildContent(task));
        pack();
        setResizable(false);
        setLocationRelativeTo(owner);
        getRootPane().setDefaultButton(closeButton);

        var im = getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
        var am = getRootPane().getActionMap();
        im.put(KeyStroke.getKeyStroke("ESCAPE"), "close");
        am.put("close", new AbstractAction() {
            @Override public void actionPerformed(ActionEvent e) { dispose(); }
        });
    }
    private JComponent buildContent(ITask t) {
        final RoundedPanel root = new RoundedPanel(AppTheme.PANEL_BG, AppTheme.TB_CORNER_RADIUS);
        root.setBorder(BorderFactory.createEmptyBorder(16, 16, 16, 16));
        root.setLayout(new BorderLayout(0, 12));

        Icon taskIcon = UiUtils.loadRasterIcon("/taskmanagement/ui/resources/task.png", 40, 40);

        JLabel title = new JLabel(ellipsize(t.getTitle(), 60));
        title.setFont(new Font("Segoe UI", Font.BOLD, 18));
        title.setForeground(AppTheme.MAIN_TEXT);

        JLabel subtitle = new JLabel("Task #" + t.getId());
        subtitle.setFont(new Font("Segoe UI", Font.PLAIN, 13));
        subtitle.setForeground(new Color(0x888888));

        JPanel titles = new JPanel();
        titles.setOpaque(false);
        titles.setLayout(new BoxLayout(titles, BoxLayout.Y_AXIS));
        titles.add(title);
        titles.add(Box.createVerticalStrut(4));
        titles.add(subtitle);

        JPanel header = new JPanel(new BorderLayout(10, 0));
        header.setOpaque(false);
        if (taskIcon != null) header.add(new JLabel(taskIcon), BorderLayout.WEST);
        header.add(titles, BorderLayout.CENTER);

        JPanel body = new JPanel();
        body.setOpaque(false);
        body.setLayout(new GridBagLayout());
        GridBagConstraints g = new GridBagConstraints();
        g.gridx = 0; g.gridy = 0;
        g.insets = new Insets(6, 4, 6, 8);
        g.anchor = GridBagConstraints.NORTHWEST;

        g.gridy++; body.add(dim("Status:"), g);
        g.gridx = 1; body.add(pill(t.getState().name()), g);

        g.gridx = 0; g.gridy++; body.add(dim("Description:"), g);
        g.gridx = 1; g.fill = GridBagConstraints.HORIZONTAL; g.weightx = 1.0;
        JTextArea ta = new JTextArea(t.getDescription() == null ? "" : t.getDescription());
        ta.setEditable(false);
        ta.setLineWrap(true);
        ta.setWrapStyleWord(true);
        ta.setBackground(new Color(60, 60, 60));
        ta.setForeground(new Color(235, 235, 235));
        ta.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        ta.setBorder(new EmptyBorder(8, 8, 8, 8));
        ta.setFocusable(false);
        ta.setHighlighter(null);
        ta.setDragEnabled(false);
        ta.setCursor(Cursor.getDefaultCursor());

        JScrollPane sp = new JScrollPane(ta,
                ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        sp.setPreferredSize(new Dimension(420, 160));
        body.add(sp, g);
```

TaskDetailsDialog

```java
        JPanel actions = new JPanel(new FlowLayout(FlowLayout.RIGHT, 8, 0));
        actions.setOpaque(false);

        closeButton = new JButton("Close");
        closeButton.setMnemonic('C');
        closeButton.getAccessibleContext().setAccessibleName("Close");
        UiUtils.styleStableHoverButton(closeButton, new Color(0x2F7BFF), Color.WHITE);
        closeButton.addActionListener(e -> dispose());
        actions.add(closeButton);

        root.add(header, BorderLayout.NORTH);
        root.add(body, BorderLayout.CENTER);
        root.add(actions, BorderLayout.SOUTH);
        return root;
    }

    private static JLabel dim(String s) {
        JLabel l = new JLabel(s);
        l.setForeground(AppTheme.MAIN_TEXT);
        l.setFont(new Font("Segoe UI", Font.BOLD, 12));
        return l;
    }

    private static JLabel val(String s) {
        JLabel l = new JLabel(s == null ? "" : s);
        l.setForeground(new Color(235, 235, 235));
        l.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        return l;
    }

    private static JLabel pill(String status) {
        String s = status == null ? "" : status;
        JLabel l = new JLabel(s, SwingConstants.CENTER);
        l.setOpaque(true);
        l.setBorder(BorderFactory.createEmptyBorder(2, 8, 2, 8));
        Color bg, fg;
        switch (s.toLowerCase()) {
            case "to-do", "todo" -> { bg = new Color(0xE74C3C); fg = Color.WHITE; }
            case "in-progress", "in progress", "inprog" -> { bg = Color.WHITE; fg = new Color(0x1E1E1E); }
            case "completed", "done" -> { bg = new Color(0x154F2A); fg = Color.WHITE; }
            default -> { bg = new Color(90, 90, 90); fg = new Color(240, 240, 240); }
        }
        l.setBackground(bg);
        l.setForeground(fg);
        l.setFont(new Font("Segoe UI", Font.BOLD, 11));
        return l;
    }

    private static String ellipsize(String s, int max) {
        if (s == null) return "";
        if (s.length() <= max) return s;
        return s.substring(0, Math.max(0, max - 1)) + "…";
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\TaskEditorDialog.java

TaskEditorDialog

```java
package taskmanagement.ui.dialogs;

import taskmanagement.domain.TaskState;
import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.util.UiUtils;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.Objects;
import java.util.Optional;

/**
 * Modal dialog for creating or editing a task.
 * <p>
 * Presentation-only and MVVM-safe: collects input and returns it to the caller
 * without accessing the model or DAO.
 */
public final class TaskEditorDialog extends JDialog {

    /**
     * Mode of the editor.
     */
    public enum Mode { ADD, EDIT }

    /**
     * Initial values for edit mode.
     *
     * @param id          task identifier
     * @param title       task title
     * @param description task description
     * @param state       task state
     */
    public static record Prefill(int id, String title, String description, TaskState state) { }

    /**
     * Result returned when the user confirms.
     *
     * @param id          task identifier (may be {@code null} in add mode)
     * @param title       task title
     * @param description task description
     * @param state       task state
     */
    public static record EditorResult(Integer id, String title, String description, TaskState state) { }

    private final Mode mode;
    private final Prefill prefill;

    private final JTextField titleField      = new JTextField(28);
    private final JTextArea  descriptionArea = new JTextArea(6, 28);
    private final JComboBox<TaskState> stateCombo = new JComboBox<>(TaskState.values());

    private final JLabel descCounter = new JLabel("0 / 500");
    private boolean confirmed = false;

    /**
     * Constructs the task editor dialog.
```

TaskEditorDialog

```java
     *
     * @param owner   parent window
     * @param mode    editor mode (ADD or EDIT)
     * @param prefill initial values used only for EDIT mode
     */
    private TaskEditorDialog(Window owner, Mode mode, Prefill prefill) {
        super(owner, mode == Mode.ADD ? "Add Task" : "Edit Task", ModalityType.APPLICATION_MODAL);
        this.mode = mode;
        this.prefill = prefill;

        setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        setResizable(false);
        setContentPane(buildContent());
        pack();
        setLocationRelativeTo(owner);

        getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ESCAPE"), "cancel");
        getRootPane().getActionMap().put("cancel", new AbstractAction() {
            @Override public void actionPerformed(ActionEvent e) { onCancel(); }
        });
        getRootPane().getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW)
                .put(KeyStroke.getKeyStroke("ctrl ENTER"), "confirm");
        getRootPane().getActionMap().put("confirm", new AbstractAction() {
            @Override public void actionPerformed(ActionEvent e) { onConfirm(); }
        });
    }

    private JComponent buildContent() {
        final RoundedPanel root = new RoundedPanel(AppTheme.PANEL_BG, AppTheme.WINDOW_CORNER_ARC);
        root.setLayout(new BorderLayout(0, 12));
        root.setBorder(new EmptyBorder(16, 18, 16, 18));

        final Color accentBg = (mode == Mode.ADD) ? new Color(0x154734) : new Color(0x1F2A44);
        final Color accentFg = new Color(0xEAF2FF);

        final RoundedPanel header = new RoundedPanel(accentBg, AppTheme.WINDOW_CORNER_ARC);
        header.setLayout(new BorderLayout(10, 0));
        header.setBorder(new EmptyBorder(10, 12, 10, 12));
        header.setOpaque(true);

        Icon hdrIcon = UiUtils.loadRasterIcon(
                mode == Mode.ADD
                        ? "/taskmanagement/ui/resources/add.png"
                        : "/taskmanagement/ui/resources/edit.png",
                22, 22
        );
        if (hdrIcon != null) {
            header.add(new JLabel(hdrIcon), BorderLayout.WEST);
        }

        JLabel hdrTitle = new JLabel(mode == Mode.ADD ? "Add New Task" : "Edit Task");
        hdrTitle.setFont(new Font("Segoe UI", Font.BOLD, 16));
        hdrTitle.setForeground(accentFg);

        JLabel hdrSub = new JLabel(mode == Mode.ADD ? "Create a new task" : "Update existing task details");
        hdrSub.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        hdrSub.setForeground(new Color(0xCFE0FF));
```

TaskEditorDialog

```
        JPanel titles = new JPanel();
        titles.setOpaque(false);
        titles.setLayout(new BoxLayout(titles, BoxLayout.Y_AXIS));
        titles.add(hdrTitle);
        titles.add(Box.createVerticalStrut(2));
        titles.add(hdrSub);
        header.add(titles, BorderLayout.CENTER);

        JPanel form = new JPanel(new GridBagLayout());
        form.setOpaque(false);
        GridBagConstraints gc = new GridBagConstraints();
        gc.insets = new Insets(6, 6, 6, 6);
        gc.fill = GridBagConstraints.HORIZONTAL;
        gc.gridx = 0; gc.gridy = 0; gc.weightx = 0;

        JLabel titleLbl = new JLabel("Title:");
        titleLbl.setForeground(AppTheme.MAIN_TEXT);
        form.add(titleLbl, gc);

        gc.gridx = 1; gc.weightx = 1.0;
        UiUtils.styleTextFieldForDarkCentered(titleField);
        titleField.setHorizontalAlignment(SwingConstants.LEFT);
        form.add(titleField, gc);

        gc.gridx = 0; gc.gridy++; gc.weightx = 0;
        JLabel descLbl = new JLabel("Description:");
        descLbl.setForeground(AppTheme.MAIN_TEXT);
        form.add(descLbl, gc);

        gc.gridx = 1; gc.weightx = 1.0;
        UiUtils.styleTextArea(descriptionArea);
        descriptionArea.setLineWrap(true);
        descriptionArea.setWrapStyleWord(true);
        descriptionArea.getDocument().addDocumentListener((UiUtils.simpleDocListener(e -> {
            int len = descriptionArea.getText().length();
            if (len > 500) {
                descriptionArea.setText(descriptionArea.getText().substring(0, 500));
                len = 500;
            }
            descCounter.setText(len + " / 500");
        })));
        JScrollPane sp = new JScrollPane(descriptionArea);
        sp.setBorder(BorderFactory.createEmptyBorder());
        sp.setOpaque(false);
        sp.getViewport().setOpaque(false);
        form.add(sp, gc);

        gc.gridx = 1; gc.gridy++; gc.weightx = 1.0;
        descCounter.setForeground(new Color(0x8CA0B3));
        descCounter.setFont(new Font("Segoe UI", Font.PLAIN, 10));
        descCounter.setHorizontalAlignment(SwingConstants.RIGHT);
        form.add(descCounter, gc);

        gc.gridx = 0; gc.gridy++; gc.weightx = 0;
        JLabel stateLbl = new JLabel("State:");
        stateLbl.setForeground(AppTheme.MAIN_TEXT);
        form.add(stateLbl, gc);

        gc.gridx = 1; gc.weightx = 1.0;
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\TaskEditorDialog.java

TaskEditorDialog

```java
        stateCombo.setFont(new Font("Segoe UI", Font.PLAIN, 12));
        form.add(stateCombo, gc);

        if (mode == Mode.EDIT && prefill != null) {
            titleField.setText(Objects.toString(prefill.title(), ""));
            descriptionArea.setText(Objects.toString(prefill.description(), ""));
            stateCombo.setSelectedItem(prefill.state());
            descCounter.setText(Math.min(descriptionArea.getText().length(), 500) + " / 500");
        } else {
            stateCombo.setSelectedItem(TaskState.ToDo);
        }

        JPanel actions = new JPanel(new FlowLayout(FlowLayout.RIGHT, 10, 0));
        actions.setOpaque(false);

        final JButton cancelButton = new JButton("Cancel");
        UiUtils.styleStableHoverButton(cancelButton, new Color(0x3B3B3B), AppTheme.MAIN_TEXT);
        cancelButton.addActionListener(e -> onCancel());

        final JButton okButton = new JButton(mode == Mode.ADD ? "Add" : "Save");
        Color primaryBg = (mode == Mode.ADD) ? new Color(0x2E8B57) : new Color(0x2F7BFF);
        UiUtils.styleStableHoverButton(okButton, primaryBg, Color.WHITE);
        okButton.addActionListener(e -> onConfirm());

        actions.add(cancelButton);
        actions.add(okButton);

        root.add(header, BorderLayout.NORTH);
        root.add(form, BorderLayout.CENTER);
        root.add(actions, BorderLayout.SOUTH);

        getRootPane().setDefaultButton(okButton);
        titleField.requestFocusInWindow();

        return root;
    }

    private void onConfirm() {
        final String title = titleField.getText().trim();
        if (title.isEmpty()) {
            JOptionPane.showMessageDialog(
                    this,
                    "Title cannot be empty.",
                    "Validation Error",
                    JOptionPane.WARNING_MESSAGE
            );
            titleField.requestFocusInWindow();
            return;
        }
        confirmed = true;
        dispose();
    }

    private void onCancel() {
        confirmed = false;
        dispose();
    }

    /**
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\dialogs\TaskEditorDialog.java

TaskEditorDialog

```java
     * Displays the dialog modally and returns the user input if confirmed.
     *
     * @param parent  parent component (for centering)
     * @param mode    dialog mode (ADD or EDIT)
     * @param prefill optional prefilled values for EDIT mode
     * @return an {@link Optional} containing {@link EditorResult} if the user confirmed; otherwise empty
     */
    public static Optional<EditorResult> showDialog(Component parent, Mode mode, Prefill prefill) {
        Window owner = (parent instanceof Window) ? (Window) parent : SwingUtilities.getWindowAncestor(parent);
        TaskEditorDialog dlg = new TaskEditorDialog(owner, mode, prefill);
        dlg.setVisible(true);

        if (!dlg.confirmed) {
            return Optional.empty();
        }
        return Optional.of(new EditorResult(
                dlg.prefill != null ? dlg.prefill.id() : null,
                dlg.titleField.getText().trim(),
                dlg.descriptionArea.getText().trim(),
                (TaskState) dlg.stateCombo.getSelectedItem()
        ));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\MainFrame.java

MainFrame

```java
package taskmanagement.ui;

import com.formdev.flatlaf.themes.FlatMacDarkLaf;
import taskmanagement.application.viewmodel.TasksViewModel;
import taskmanagement.persistence.DAOProvider;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.ui.api.TasksViewAPI;
import taskmanagement.ui.adapters.TasksViewApiAdapter;
import taskmanagement.ui.chrome.WindowChrome;
import taskmanagement.ui.dialogs.AboutDialog;
import taskmanagement.ui.dialogs.ConfirmExitDialog;
import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.UiUtils;
import taskmanagement.ui.views.ContentArea;
import taskmanagement.ui.widgets.HeaderBar;

import javax.swing.*;
import java.awt.*;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.awt.event.WindowEvent;

/**
 * Borderless main application window that initializes Look & Feel, wires the MVVM stack
 * (DAO → ViewModel → API), and hosts the root UI including header and content area.
 * <p>
 * This class contains window-level concerns only; no domain logic is implemented here.
 * </p>
 */
public final class MainFrame extends JFrame {

    private HeaderBar header;
    private final TasksViewAPI api;

    /**
     * Constructs the main frame, sets the look and feel, prepares DAO → ViewModel → API wiring,
     * and builds the UI hierarchy.
     */
    public MainFrame() {
        super("Task Management App");
        try {
            UIManager.setLookAndFeel(new FlatMacDarkLaf());
        } catch (Throwable ignore) {
        }

        ImageIcon appIcon = (ImageIcon) UiUtils.loadRasterIcon(
                "/taskmanagement/ui/resources/tasks_mng.png", 64, 64);
        if (appIcon != null) {
            setIconImage(appIcon.getImage());
        }

        ITasksDAO dao = DAOProvider.get();
        TasksViewModel vm = new TasksViewModel(dao);
        this.api = new TasksViewApiAdapter(vm);

        setContentPane(buildRoot());
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);

        WindowChrome.makeBorderlessWithRoundedCorners(this);
        setPreferredSize(new Dimension(AppTheme.APP_WIDTH, AppTheme.APP_HEIGHT));
        setBackground(AppTheme.APP_BG);
        pack();
        setLocationRelativeTo(null);
    }

    private JComponent buildRoot() {
        JPanel root = new JPanel(new BorderLayout());
        root.setOpaque(true);
        root.setBackground(AppTheme.APP_BG);

        header = new HeaderBar();
        header.setTitleText("Task Management App");
        header.onAbout(btn -> AboutDialog.showDialog(this));
        header.onClose(btn -> {
            boolean ok = ConfirmExitDialog.confirm(this);
            if (ok) {
                dispatchEvent(new WindowEvent(MainFrame.this, WindowEvent.WINDOW_CLOSING));
            }
        });

        installDragOn(header);
        try {
            var m = WindowChrome.class.getDeclaredMethod("installDragHandler", JFrame.class, JComponent.class);
            m.invoke(null, this, header);
        } catch (Throwable ignored) {
        }

        JPanel headerWrap = new JPanel(new BorderLayout());
        headerWrap.setOpaque(false);
        headerWrap.setBorder(BorderFactory.createEmptyBorder(
                AppTheme.PADDING, AppTheme.PADDING, 0, AppTheme.PADDING));
        headerWrap.add(header, BorderLayout.CENTER);
        root.add(headerWrap, BorderLayout.NORTH);

        JPanel bodyBackground = new JPanel(new BorderLayout());
        bodyBackground.setOpaque(true);
        bodyBackground.setBackground(AppTheme.BODY_BG);
        bodyBackground.setBorder(BorderFactory.createEmptyBorder(
                AppTheme.PADDING, AppTheme.PADDING, AppTheme.PADDING, AppTheme.PADDING));
        root.add(bodyBackground, BorderLayout.CENTER);

        ContentArea content = new ContentArea();
        content.setApi(api);
        api.reload();
        bodyBackground.add(content, BorderLayout.CENTER);

        installEscToClose();
        return root;
    }

    private void installEscToClose() {
        JRootPane root = getRootPane();
        InputMap im = root.getInputMap(JComponent.WHEN_IN_FOCUSED_WINDOW);
        ActionMap am = root.getActionMap();
        final String key = "app-close";
        im.put(KeyStroke.getKeyStroke(java.awt.event.KeyEvent.VK_ESCAPE, 0), key);
        am.put(key, new AbstractAction() {
```

MainFrame

```java
            @Override
            public void actionPerformed(java.awt.event.ActionEvent e) {
                boolean ok = ConfirmExitDialog.confirm(MainFrame.this);
                if (ok) {
                    dispatchEvent(new WindowEvent(MainFrame.this, WindowEvent.WINDOW_CLOSING));
                }
            }
        });
    }

    private void installDragOn(JComponent handle) {
        final Point[] origin = new Point[1];
        MouseAdapter ma = new MouseAdapter() {
            @Override public void mousePressed(MouseEvent e) { origin[0] = e.getPoint(); }
            @Override public void mouseDragged(MouseEvent e) {
                if (origin[0] != null) {
                    Point p = e.getLocationOnScreen();
                    Insets ins = getInsets();
                    setLocation(p.x - origin[0].x - ins.left, p.y - origin[0].y - ins.top);
                }
            }
            @Override public void mouseReleased(MouseEvent e) { origin[0] = null; }
        };
        handle.addMouseListener(ma);
        handle.addMouseMotionListener(ma);
    }

    /**
     * Launches the application window.
     *
     * @param args ignored
     */
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            MainFrame f = new MainFrame();
            f.setVisible(true);
        });
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\styles\AppTheme.java

AppTheme

```java
package taskmanagement.ui.styles;

import javax.swing.*;
import javax.swing.plaf.ComboBoxUI;
import javax.swing.plaf.basic.BasicComboBoxUI;
import java.awt.*;

/**
 * Provides centralized, immutable theme tokens for the application UI, including
 * dimensions, colors, and fonts. This class is a constant holder and cannot be instantiated.
 */
public final class AppTheme {

    private AppTheme() { }

    /** Logical application width in pixels (initial/suggested). */
    public static final int APP_WIDTH = 1000;

    /** Logical application height in pixels (initial/suggested). */
    public static final int APP_HEIGHT = 640;

    /** Corner arc radius (px) for rounded windows/containers. */
    public static final int WINDOW_CORNER_ARC = 24;

    /** App background color (dark). */
    public static final Color APP_BG = new Color(0x121212);

    /** Body/content background color (dark). */
    public static final Color BODY_BG = new Color(0x121212);

    /** Header/background color for top bars and cards. */
    public static final Color HEADER_BG = new Color(0x1E1E1E);

    /** Global primary accent color for titles/emphasis. */
    public static final Color ACCENT_PRIMARY = Color.BLACK;

    /** Secondary accent color. */
    public static final Color ACCENT_SECONDARY = new Color(0xFFFFFF);

    /** Main application title color. */
    public static final Color MAIN_APP_TITLE = new Color(0xFFE690);

    /**
     * Legacy accent color kept for backward compatibility; prefer {@link #ACCENT_PRIMARY}.
     * @deprecated Use {@link #ACCENT_PRIMARY} instead.
     */
    @Deprecated public static final Color IOS_ORANGE = new Color(0xFFA100);

    /** Accent/danger color (iOS-like red). */
    public static final Color IOS_RED = new Color(0x630700);

    /** Light cream color for pills and highlights. */
    public static final Color CREAM_WHITE = new Color(0xFAFAE1);

    /** Neutral dark grey for panels and rails. */
    public static final Color DARK_GREY = new Color(0x2B2B2B);

    /** Primary foreground text color on dark backgrounds. */
    public static final Color MAIN_TEXT = new Color(0xFFFFFF);
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\styles\AppTheme.java

AppTheme

```java
    /** Generic padding unit (px) for containers. */
    public static final int PADDING = 12;

    /** Horizontal padding (px) inside header areas. */
    public static final int HEADER_HPAD = 8;

    /** Horizontal gap (px) between action buttons. */
    public static final int ACTIONS_HGAP = 28;

    /** Vertical gap (px) between stacked action buttons. */
    public static final int ACTIONS_VGAP = 8;

    /** Default corner radius (px) for rounded buttons. */
    public static final int BTN_RADIUS = 12;

    /** Vertical inner padding (px) for generic buttons. */
    public static final int BTN_PAD_V = 10;

    /** Horizontal inner padding (px) for generic buttons. */
    public static final int BTN_PAD_H = 18;

    /** Default font size (pt) for generic buttons. */
    public static final int BTN_FONT = 14;

    /** Title font size (pt) for large headings. */
    public static final int TITLE_FONT = 36;

    /** Add button background color. */
    public static final Color CTRL_ADD_BG = new Color(144, 236, 152);

    /** Edit button background color. */
    public static final Color CTRL_EDIT_BG = new Color(255, 203, 160);

    /** Delete button background color. */
    public static final Color CTRL_DELETE_BG = new Color(248, 130, 130);

    /** Add button foreground color. */
    public static final Color CTRL_ADD_FG = new Color(0, 50, 4);

    /** Edit button foreground color. */
    public static final Color CTRL_EDIT_FG = new Color(115, 49, 0);

    /** Delete button foreground color. */
    public static final Color CTRL_DELETE_FG = new Color(83, 0, 0);

    /** Refresh button background color (teal). */
    public static final Color CTRL_REFRESH_BG = new Color(0x2EC4B6);

    /** Refresh button foreground color. */
    public static final Color CTRL_REFRESH_FG = new Color(0x003E36);

    /** Disk Cleanup button background color (deep lilac). */
    public static final Color CTRL_CLEANUP_BG = new Color(0x6C5CE7);

    /** Disk Cleanup button foreground color. */
    public static final Color CTRL_CLEANUP_FG = new Color(0xC2BDF3);

    /** Neutral foreground for controls placed on dark backgrounds. */
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\styles\AppTheme.java

AppTheme

```java
    public static final Color CTRL_ON_DARK_FG = MAIN_TEXT;

    /** Square control block size (px) used for rail buttons. */
    public static final int CTRL_BLOCK_SIZE = 80;

    /** Icon size (px) inside a control block. */
    public static final int CTRL_ICON_SIZE = 25;

    /** Corner radius (px) for control blocks. */
    public static final int CTRL_CORNER_RAD = 12;

    /** Label font size (pt) under control icons. */
    public static final float CTRL_FONT_SIZE = 12f;

    /** Default bold font for control buttons. */
    public static final Font CTRL_BUTTON_FONT = new Font("Segoe UI", Font.BOLD, 12);

    /** Uniform padding (px) inside toolbox containers. */
    public static final int TB_PAD = 8;

    /** Vertical gap (px) between toolbox rows. */
    public static final int TB_GAP = 8;

    /** Smaller gap for tight rows. */
    public static final int TB_GAP_SM = 6;

    /** Corner radius (px) for toolbox rounded containers. */
    public static final int TB_CORNER_RADIUS = 12;

    /** Toolbox large title font. */
    public static final Font TB_TITLE_FONT_LG = new Font("Segoe UI", Font.BOLD, 16);

    /** Toolbox label font. */
    public static final Font TB_LABEL_FONT_LG = new Font("Segoe UI", Font.PLAIN, 14);

    /** Toolbox radio/checkbox font. */
    public static final Font TB_RADIO_FONT = new Font("Segoe UI", Font.PLAIN, 13);

    /** Preferred field width (px) to keep right column near ~20%. */
    public static final int TB_FIELD_WIDTH = 160;

    /** Preferred field height (px). */
    public static final int TB_FIELD_HEIGHT = 34;

    /** Toolbox foreground text color. */
    public static final Color TB_TEXT_FG = MAIN_TEXT;

    /** Toolbox field background color. */
    public static final Color TB_FIELD_BG = new Color(0x30, 0x30, 0x30);

    /** Toolbox field border color. */
    public static final Color TB_FIELD_BORDER = new Color(0x4A, 0x4A, 0x4A);

    /** Neutral panel background color (used by rounded panels). */
    public static final Color PANEL_BG = DARK_GREY;

    /** Export button preferred width (px). */
    public static final int TB_EXPORT_W = 150;
```

AppTheme

```
    /** Export button preferred height (px). */
    public static final int TB_EXPORT_H = 70;

    /** Export button corner radius (px). */
    public static final int TB_EXPORT_RADIUS = 12;

    /** Export button icon size (px). */
    public static final int TB_EXPORT_ICON = 35;

    /** Export button font. */
    public static final Font TB_EXPORT_FONT = new Font("Segoe UI", Font.BOLD, 17);

    /** Export button background color. */
    public static final Color TB_EXPORT_BG = new Color(136, 198, 228);

    /** Export button foreground color. */
    public static final Color TB_EXPORT_FG = new Color(14, 26, 76);

    /** Undo button background color. */
    public static final Color TB_UNDO_BG = new Color(0x3A3A3A);

    /** Undo button foreground color. */
    public static final Color TB_UNDO_FG = MAIN_TEXT;

    /** Redo button background color. */
    public static final Color TB_REDO_BG = new Color(0x3A3A3A);

    /** Redo button foreground color. */
    public static final Color TB_REDO_FG = MAIN_TEXT;

    /** Advance button background color. */
    public static final Color TB_ADVANCE_BG = new Color(0x2F3B26);

    /** Advance button foreground color. */
    public static final Color TB_ADVANCE_FG = new Color(0xCFF5C0);

    /** Mark-as button background color. */
    public static final Color TB_MARK_BG = new Color(0x2B3442);

    /** Mark-as button foreground color. */
    public static final Color TB_MARK_FG = new Color(0xC8E3FF);

    /** Generic icon size (px) for ToolBox action buttons. */
    public static final int TB_ACTION_ICON = 28;

    /** Sort Apply button background color. */
    public static final Color TB_SORT_APPLY_BG = new Color(0x3C5D2A);

    /** Sort Apply button foreground color. */
    public static final Color TB_SORT_APPLY_FG = new Color(0xD9F7C6);

    /** Sort Reset button background color. */
    public static final Color TB_SORT_RESET_BG = new Color(0x5D2A2A);

    /** Sort Reset button foreground color. */
    public static final Color TB_SORT_RESET_FG = new Color(0xFAD4D4);

    /** Filter Apply button background color. */
    public static final Color TB_FILTER_APPLY_BG = new Color(0x3C5D2A);
```

AppTheme

```java
    /** Filter Apply button foreground color. */
    public static final Color TB_FILTER_APPLY_FG = new Color(0xD9F7C6);

    /** Filter Reset button background color. */
    public static final Color TB_FILTER_RESET_BG = new Color(0x5D2A2A);

    /** Filter Reset button foreground color. */
    public static final Color TB_FILTER_RESET_FG = new Color(0xFAD4D4);

    /** Show-Filtered toggle border color (OFF state). */
    public static final Color TB_SHOW_BORDER = TB_FIELD_BORDER;

    /** Show-Filtered toggle foreground color (OFF state). */
    public static final Color TB_SHOW_FG = TB_TEXT_FG;

    /** Show-Filtered toggle background color (ON state). */
    public static final Color TB_SHOW_SELECTED_BG = new Color(0x2B3E5A);

    /** Show-Filtered toggle foreground color (ON state). */
    public static final Color TB_SHOW_SELECTED_FG = new Color(0xD6E8FF);

    /** Header button fixed height (px). */
    public static final int HB_BTN_HEIGHT = 38;

    /** Header button minimum width (px). */
    public static final int HB_BTN_MIN_W = 110;

    /** Header button corner radius (px). */
    public static final int HB_BTN_RADIUS = 12;

    /** Header button font. */
    public static final Font HB_BTN_FONT = new Font("Segoe UI", Font.BOLD, 14);

    /** About button background color. */
    public static final Color HB_ABOUT_BG = CREAM_WHITE;

    /** About button foreground color. */
    public static final Color HB_ABOUT_FG = Color.BLACK;

    /** Close button background color. */
    public static final Color HB_CLOSE_BG = IOS_RED;

    /** Close button foreground color. */
    public static final Color HB_CLOSE_FG = Color.WHITE;

    /**
     * Applies global Swing defaults to remove the default OS blue accent in common components.
     * This method should be invoked on the Event Dispatch Thread (EDT) before creating components.
     */
    public static void applyAccentDefaults() {
        UIManager.put("List.selectionBackground", SELECTION_BG);
        UIManager.put("List.selectionForeground", SELECTION_FG);
        UIManager.put("Table.selectionBackground", SELECTION_BG);
        UIManager.put("Table.selectionForeground", SELECTION_FG);
        UIManager.put("Tree.selectionBackground", SELECTION_BG);
        UIManager.put("Tree.selectionForeground", SELECTION_FG);
        UIManager.put("TextField.selectionBackground", SELECTION_BG);
        UIManager.put("TextField.selectionForeground", SELECTION_FG);
        UIManager.put("TextArea.selectionBackground", SELECTION_BG);
        UIManager.put("TextArea.selectionForeground", SELECTION_FG);
        UIManager.put("ComboBox.selectionBackground", SELECTION_BG);
        UIManager.put("ComboBox.selectionForeground", SELECTION_FG);
        UIManager.put("ComboBox.background", TB_FIELD_BG);
        UIManager.put("ComboBox.foreground", TB_TEXT_FG);
        UIManager.put("ComboBox.buttonBackground", TB_FIELD_BG);
        UIManager.put("ComboBox.buttonShadow", ACCENT_PRIMARY);
        UIManager.put("ComboBox.buttonDarkShadow", ACCENT_PRIMARY);
        UIManager.put("CheckBox.icon", new AccentCheckBoxIcon());
    }

    /** Selection background color used across lists, popups, and fields. */
    public static final Color SELECTION_BG = new Color(0x2B2B2B);

    /** Selection foreground color used across lists, popups, and fields. */
    public static final Color SELECTION_FG = MAIN_TEXT;

    /**
     * Provides a flat {@link JComboBox} UI with a neutral arrow and no bright accent color.
     * Intended to be applied as: {@code combo.setUI(AppTheme.flatComboUI());}
     *
     * @return a {@link ComboBoxUI} instance with a flat arrow button and themed colors
     */
    public static ComboBoxUI flatComboUI() {
        return new BasicComboBoxUI() {
            @Override
            protected JButton createArrowButton() {
                JButton b = new JButton("▾");
                b.setBorder(BorderFactory.createEmptyBorder(2, 8, 2, 8));
                b.setFocusable(false);
                b.setOpaque(true);
                b.setBackground(TB_FIELD_BG);
                b.setForeground(ACCENT_PRIMARY);
                return b;
            }
        };
    }

    /**
     * Minimal accent-colored checkbox icon used to override the default OS check style.
     * The icon respects the themed background, border, and selected state.
     */
    public static final class AccentCheckBoxIcon implements Icon {

        private static final int SZ = 18, ARC = 4;

        /**
         * Returns the icon width in pixels.
         *
         * @return the icon width
         */
        @Override
        public int getIconWidth() {
            return SZ;
        }

        /**
         * Returns the icon height in pixels.
         *
         * @return the icon height
         */
        @Override
        public int getIconHeight() {
            return SZ;
        }

        /**
         * Paints the icon at the specified location.
         *
         * @param c the component to which the icon is painted
         * @param g the graphics context
         * @param x the X coordinate of the icon's top-left corner
         * @param y the Y coordinate of the icon's top-left corner
         */
        @Override
        public void paintIcon(Component c, Graphics g, int x, int y) {
            AbstractButton b = (AbstractButton) c;
            Graphics2D g2 = (Graphics2D) g.create();
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);

            g2.setColor(TB_FIELD_BG);
            g2.fillRoundRect(x, y, SZ, SZ, ARC, ARC);
            g2.setColor(TB_FIELD_BORDER);
            g2.drawRoundRect(x, y, SZ, SZ, ARC, ARC);

            if (b.isSelected()) {
                g2.setStroke(new BasicStroke(2.2f, BasicStroke.CAP_ROUND, BasicStroke.JOIN_ROUND));
                g2.setColor(ACCENT_SECONDARY);
                int x1 = x + 4,  y1 = y + 9;
                int x2 = x + 8,  y2 = y + 13;
                int x3 = x + 14, y3 = y + 5;
                g2.drawLine(x1, y1, x2, y2);
                g2.drawLine(x2, y2, x3, y3);
            }
            g2.dispose();
        }
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\util\RoundedPanel.java

RoundedPanel

```java
package taskmanagement.ui.util;

import javax.swing.*;
import java.awt.*;

/**
 * A custom {@link JPanel} with rounded corners and a configurable background color.
 * <p>
 * This panel disables default opacity so that its rounded edges
 * can blend seamlessly with the parent background.
 * </p>
 */
public class RoundedPanel extends JPanel {

    /** Radius (in pixels) used for drawing rounded corners. */
    private final int arc;

    /** Background color used to fill the rounded panel. */
    private final Color bg;

    /**
     * Creates a new rounded panel with the specified background color and corner radius.
     *
     * @param bg  the background color of the panel
     * @param arc the radius (in pixels) for the rounded corners
     */
    public RoundedPanel(Color bg, int arc) {
        super();
        this.bg = bg;
        this.arc = arc;
        setOpaque(false); // ensure background outside rounded area is transparent
    }

    /**
     * Paints the panel with rounded corners using the configured background color.
     *
     * @param g the {@link Graphics} context used for painting
     */
    @Override
    protected void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D) g.create();
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(bg);
        g2.fillRoundRect(0, 0, getWidth(), getHeight(), arc, arc);
        g2.dispose();
        super.paintComponent(g);
    }
}
```

UiUtils

```java
package taskmanagement.ui.util;

import taskmanagement.ui.styles.AppTheme;

import javax.swing.*;
import javax.swing.border.Border;
import java.awt.*;

import javax.swing.event.DocumentEvent;
import javax.swing.event.DocumentListener;
import java.util.function.Consumer;

/**
 * Swing utilities for styling widgets in a consistent, theme-aware way.
 * <p>
 * This class is append-only: legacy helpers remain intact and new helpers
 * are added without breaking existing signatures.
 * </p>
 */
public final class UiUtils {
    private UiUtils() {}

    /**
     * Styles a {@link JButton} to keep stable size on hover/press (no layout shift)
     * while providing simple color feedback.
     *
     * @param b       the button to style
     * @param baseBg base background color
     * @param baseFg base foreground (text/icon) color
     */
    public static void styleStableHoverButton(JButton b, Color baseBg, Color baseFg) {
        b.setFocusPainted(false);
        b.setRolloverEnabled(true);
        b.setContentAreaFilled(true);
        b.setOpaque(true);
        b.setBackground(baseBg);
        b.setForeground(baseFg);
        b.setFont(b.getFont().deriveFont(Font.BOLD, (float) AppTheme.BTN_FONT));
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        Border constantBorder = BorderFactory.createCompoundBorder(
                new RoundedMatteShadowBorder(new Color(0, 0, 0, 60), AppTheme.BTN_RADIUS, 1),
                BorderFactory.createEmptyBorder(AppTheme.BTN_PAD_V, AppTheme.BTN_PAD_H,
                        AppTheme.BTN_PAD_V, AppTheme.BTN_PAD_H)
        );
        b.setBorder(constantBorder);
        final Color hoverBg = shiftForContrast(baseBg, 0.40f);
        final Color hoverFg = bestTextFor(hoverBg, baseFg);
        final Color pressBg = shiftForContrast(baseBg, 0.55f);
        final Color pressFg = bestTextFor(pressBg, baseFg);
        b.getModel().addChangeListener(e -> {
            ButtonModel m = (ButtonModel) e.getSource();
            if (m.isPressed()) {
                b.setBackground(pressBg);
                b.setForeground(pressFg);
            } else if (m.isRollover()) {
                b.setBackground(hoverBg);
                b.setForeground(hoverFg);
            } else {
                b.setBackground(baseBg);
                b.setForeground(baseFg);
            }
        });
    }

    /**
     * Shifts a color toward light or dark depending on its luminance.
     *
     * @param c     the base color
     * @param ratio blend ratio in [0..1]
     * @return a color blended toward black or white to increase contrast
     */
    public static Color shiftForContrast(Color c, float ratio) {
        float lum = luminance(c);
        return (lum >= 0.5f) ? blend(c, Color.BLACK, ratio) : blend(c, Color.WHITE, ratio);
    }

    /**
     * Chooses a readable text color for a given background.
     *
     * @param bg        background color
     * @param preferred preferred foreground
     * @return the most readable color among preferred/white/black
     */
    public static Color bestTextFor(Color bg, Color preferred) {
        if (contrastRatio(bg, preferred) >= 4.0) return preferred;
        Color alt1 = Color.WHITE, alt2 = Color.BLACK;
        double bestC = contrastRatio(bg, preferred);
        Color best = preferred;
        double c1 = contrastRatio(bg, alt1);
        if (c1 > bestC) { best = alt1; bestC = c1; }
        double c2 = contrastRatio(bg, alt2);
        if (c2 > bestC) { best = alt2; }
        return best;
    }

    /**
     * Linear blend between two colors.
     *
     * @param a first color
     * @param b second color
     * @param t blend factor in [0..1]
     * @return blended color
     */
    public static Color blend(Color a, Color b, float t) {
        t = Math.max(0f, Math.min(1f, t));
        int r = Math.round(a.getRed() + (b.getRed() - a.getRed()) * t);
        int g = Math.round(a.getGreen() + (b.getGreen() - a.getGreen()) * t);
        int bl = Math.round(a.getBlue() + (b.getBlue() - a.getBlue()) * t);
        int al = Math.round(a.getAlpha() + (b.getAlpha() - a.getAlpha()) * t);
        return new Color(r, g, bl, al);
    }

    /**
     * Approximates sRGB relative luminance.
     *
     * @param c color
     * @return luminance in [0..1]
     */
```

UiUtils

```java
    public static float luminance(Color c) {
        float r = srgbToLin(c.getRed() / 255f);
        float g = srgbToLin(c.getGreen() / 255f);
        float b = srgbToLin(c.getBlue() / 255f);
        return (float) (0.2126 * r + 0.7152 * g + 0.0722 * b);
    }

    /**
     * Converts sRGB component to linear space.
     *
     * @param c component value in [0..1]
     * @return linearized component
     */
    public static float srgbToLin(float c) {
        return (c <= 0.04045f) ? (c / 12.92f) : (float) Math.pow((c + 0.055f) / 1.055f, 2.4);
    }

    /**
     * Computes a WCAG-like contrast ratio.
     *
     * @param a first color
     * @param b second color
     * @return contrast ratio (>=1)
     */
    public static double contrastRatio(Color a, Color b) {
        double la = luminance(a) + 0.05;
        double lb = luminance(b) + 0.05;
        return (Math.max(la, lb) / Math.min(la, lb));
    }

    /**
     * Rounded matte border with a soft bottom shadow (no external libraries).
     */
    public static final class RoundedMatteShadowBorder extends javax.swing.border.AbstractBorder {
        private final Color shadow;
        private final int arc;
        private final int depth;

        /**
         * Creates a new rounded border with a matte shadow at the bottom edge.
         *
         * @param shadow shadow color
         * @param arc    corner arc radius
         * @param depth  shadow thickness in pixels (min 1)
         */
        public RoundedMatteShadowBorder(Color shadow, int arc, int depth) {
            this.shadow = shadow; this.arc = arc; this.depth = Math.max(1, depth);
        }

        /** {@inheritDoc} */
        @Override public Insets getBorderInsets(Component c) { return new Insets(4, 8, 4 + depth, 8); }

        /** {@inheritDoc} */
        @Override public boolean isBorderOpaque() { return false; }

        /** {@inheritDoc} */
        @Override
        public void paintBorder(Component c, Graphics g, int x, int y, int w, int h) {
            Graphics2D g2 = (Graphics2D) g.create();
```

UiUtils

```java
            g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            g2.setColor(shadow);
            g2.fillRoundRect(x, y + h - depth + 1, w - 1, depth, arc, arc);
            g2.dispose();
        }
    }

    /**
     * Aligns a component horizontally to the center (useful for BoxLayout Y_AXIS stacks).
     *
     * @param c component to center
     */
    public static void centerHoriz(JComponent c) {
        c.setAlignmentX(Component.CENTER_ALIGNMENT);
    }

    /**
     * Creates a square {@link JButton} with icon (top) and text (bottom), styled for dark backgrounds.
     * Legacy signature used by ControlPanel.
     *
     * @param text         button text
     * @param icon         button icon (nullable)
     * @param bg           background color
     * @param fg           foreground color
     * @param size         square size in pixels
     * @param cornerRadius corner radius
     * @param fontSize     label font size
     * @return configured button
     */
    public static JButton createSquareActionButton(
            String text,
            Icon icon,
            Color bg,
            Color fg,
            int size,
            int cornerRadius,
            float fontSize
    ) {
        JButton b = new JButton(text);
        b.setHorizontalTextPosition(SwingConstants.CENTER);
        b.setVerticalTextPosition(SwingConstants.BOTTOM);
        b.setFocusPainted(false);
        b.setBorderPainted(false);
        b.setContentAreaFilled(true);
        b.setOpaque(true);
        b.setBackground(bg);
        b.setForeground(fg);
        b.setFont(b.getFont().deriveFont(Font.PLAIN, fontSize));
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        Dimension d = new Dimension(size, size);
        b.setPreferredSize(d);
        b.setMinimumSize(d);
        b.setMaximumSize(d);
        if (icon != null) {
            b.setIcon(icon);
        } else {
            b.setIcon(makeDotIcon(Math.max(10, Math.min(22, size / 2)), fg));
        }
        b.putClientProperty("JButton.buttonType", "roundRect");
        b.putClientProperty("JComponent.roundRect", true);
        b.putClientProperty("JComponent.arc", cornerRadius);
        addDarkHoverAndPress(b, bg);
        return b;
    }

    /**
     * Attaches hover/pressed background behavior suitable for dark themes.
     *
     * @param b    button to style
     * @param base base background color
     */
    public static void addDarkHoverAndPress(AbstractButton b, Color base) {
        final Color hover = darken(base, 0.06f);
        final Color press = darken(base, 0.12f);
        b.addChangeListener(e -> {
            ButtonModel m = b.getModel();
            if (m.isPressed()) {
                b.setBackground(press);
            } else if (m.isRollover()) {
                b.setBackground(hover);
            } else {
                b.setBackground(base);
            }
        });
    }

    /**
     * Darkens a color by a given factor.
     *
     * @param c      base color
     * @param factor darkening factor in [0..1]
     * @return darkened color
     */
    public static Color darken(Color c, float factor) {
        factor = Math.max(0f, Math.min(1f, factor));
        int r = Math.max(0, (int) (c.getRed() * (1f - factor)));
        int g = Math.max(0, (int) (c.getGreen() * (1f - factor)));
        int b = Math.max(0, (int) (c.getBlue() * (1f - factor)));
        return new Color(r, g, b, c.getAlpha());
    }

    /**
     * Loads a raster icon from the classpath and scales it.
     *
     * @param classpath resource path (e.g., "/icons/add.png")
     * @param w         target width
     * @param h         target height
     * @return an {@link Icon} or {@code null} if not found or failed
     */
    public static Icon loadRasterIcon(String classpath, int w, int h) {
        try {
            java.net.URL url = UiUtils.class.getResource(classpath);
            if (url == null) return null;
            Image img = new ImageIcon(url).getImage().getScaledInstance(w, h, Image.SCALE_SMOOTH);
            return new ImageIcon(img);
        } catch (Exception ignore) {
            return null;
        }
    }

    /**
     * Attempts to load an SVG icon using FlatLaf Extras (if present on the classpath).
     *
     * @param classpath SVG resource path
     * @param w         target width
     * @param h         target height
     * @return an {@link Icon} or {@code null} when FlatLaf Extras is unavailable
     */
    public static Icon tryLoadSvgIcon(String classpath, int w, int h) {
        try {
            Class<?> svgIconCls = Class.forName("com.formdev.flatlaf.extras.FlatSVGIcon");
            return (Icon) svgIconCls
                    .getConstructor(String.class, int.class, int.class)
                    .newInstance(classpath, w, h);
        } catch (Throwable t) {
            return null;
        }
    }

    /**
     * Creates a small fallback icon (filled circle).
     *
     * @param size  icon size
     * @param color fill color
     * @return an {@link Icon} instance
     */
    public static Icon makeDotIcon(int size, Color color) {
        return new Icon() {
            @Override public void paintIcon(Component c, Graphics g, int x, int y) {
                g.setColor(color);
                int s = Math.min(size, getIconWidth());
                g.fillOval(x + (getIconWidth() - s) / 2, y + (getIconHeight() - s) / 2, s, s);
            }
            @Override public int getIconWidth()  { return size; }
            @Override public int getIconHeight() { return size; }
        };
    }

    /**
     * Creates a {@link DocumentListener} that invokes the same {@link Runnable} for all events.
     *
     * @param r action to execute
     * @return a document listener
     */
    public static DocumentListener simpleDocListener(Runnable r) {
        return new DocumentListener() {
            @Override public void insertUpdate(DocumentEvent e)  { r.run(); }
            @Override public void removeUpdate(DocumentEvent e)  { r.run(); }
            @Override public void changedUpdate(DocumentEvent e) { r.run(); }
        };
    }

    /**
     * Creates a {@link DocumentListener} that forwards events to a {@link Consumer}.
     *
     * @param c consumer receiving each {@link DocumentEvent}
     * @return a document listener
```

UiUtils

```java
     */
    public static DocumentListener simpleDocListener(Consumer<DocumentEvent> c) {
        return new DocumentListener() {
            @Override public void insertUpdate(DocumentEvent e)  { c.accept(e); }
            @Override public void removeUpdate(DocumentEvent e)  { c.accept(e); }
            @Override public void changedUpdate(DocumentEvent e) { c.accept(e); }
        };
    }

    /**
     * Styles a {@link JTextField} for dark panels using {@link AppTheme} tokens (centered).
     *
     * @param field the text field to style
     */
    public static void styleTextFieldForDarkCentered(JTextField field) {
        field.putClientProperty("JTextField.showClearButton", true);
        field.setForeground(AppTheme.TB_TEXT_FG);
        field.setBackground(AppTheme.TB_FIELD_BG);
        field.setCaretColor(AppTheme.TB_TEXT_FG);
        field.setSelectionColor(darken(AppTheme.TB_FIELD_BG, 0.25f));
        field.setSelectedTextColor(AppTheme.TB_TEXT_FG);
        field.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(AppTheme.TB_FIELD_BORDER),
                BorderFactory.createEmptyBorder(6, 8, 6, 8)
        ));
        field.setHorizontalAlignment(SwingConstants.CENTER);
        Dimension d = new Dimension(AppTheme.TB_FIELD_WIDTH, AppTheme.TB_FIELD_HEIGHT);
        field.setPreferredSize(d);
        field.setMinimumSize(d);
    }

    /**
     * Styles a generic {@link JTextField} for dark UI (left-aligned).
     *
     * @param field the text field to style
     */
    public static void styleTextField(JTextField field) {
        styleTextField(field, AppTheme.TB_FIELD_BG, AppTheme.TB_TEXT_FG, AppTheme.TB_FIELD_BORDER, false);
    }

    /**
     * Styles a {@link JTextField} with explicit colors and alignment.
     *
     * @param field    the field to style
     * @param bg       background color
     * @param fg       text color
     * @param border   border color
     * @param centered whether to center the text horizontally
     */
    public static void styleTextField(JTextField field, Color bg, Color fg, Color border, boolean centered) {
        field.putClientProperty("JTextField.showClearButton", true);
        field.setForeground(fg);
        field.setBackground(bg);
        field.setCaretColor(fg);
        field.setSelectionColor(darken(bg, 0.25f));
        field.setSelectedTextColor(fg);
        field.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(border),
                BorderFactory.createEmptyBorder(6, 8, 6, 8)
```

UiUtils

```java
        ));
        field.setHorizontalAlignment(centered ? SwingConstants.CENTER : SwingConstants.LEADING);
    }

    /**
     * Styles a {@link JTextArea} for dark UI in dialogs (e.g., task description).
     *
     * @param area the text area to style
     */
    public static void styleTextArea(JTextArea area) {
        area.setForeground(AppTheme.TB_TEXT_FG);
        area.setBackground(AppTheme.TB_FIELD_BG);
        area.setCaretColor(AppTheme.TB_TEXT_FG);
        area.setSelectionColor(darken(AppTheme.TB_FIELD_BG, 0.25f));
        area.setSelectedTextColor(AppTheme.TB_TEXT_FG);
        area.setLineWrap(true);
        area.setWrapStyleWord(true);
        area.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(AppTheme.TB_FIELD_BORDER),
                BorderFactory.createEmptyBorder(6, 8, 6, 8)
        ));
    }

    /**
     * Styles a ToolBox title label centered.
     *
     * @param label label to style
     */
    public static void styleToolBoxTitleCentered(JLabel label) {
        label.setOpaque(false);
        label.setForeground(AppTheme.TB_TEXT_FG);
        if (AppTheme.TB_TITLE_FONT_LG != null) label.setFont(AppTheme.TB_TITLE_FONT_LG);
        label.setHorizontalAlignment(SwingConstants.CENTER);
    }

    /**
     * Styles a ToolBox regular label centered.
     *
     * @param label label to style
     */
    public static void styleToolBoxLabelCentered(JLabel label) {
        label.setOpaque(false);
        label.setForeground(AppTheme.TB_TEXT_FG);
        if (AppTheme.TB_LABEL_FONT_LG != null) label.setFont(AppTheme.TB_LABEL_FONT_LG);
        label.setHorizontalAlignment(SwingConstants.CENTER);
    }

    /**
     * Wraps a single child in a transparent {@link JPanel} with centered {@link FlowLayout}.
     *
     * @param child component to center
     * @return panel that centers the child
     */
    public static JPanel flowCenter(JComponent child) {
        JPanel p = new JPanel(new FlowLayout(FlowLayout.CENTER, 0, 0));
        p.setOpaque(false);
        p.add(child);
        return p;
    }
```

UiUtils

```java
    /**
     * Creates a primary rounded button with icon (left) and text (right),
     * painted locally to avoid Look-and-Feel artifacts.
     *
     * @param text         button text
     * @param icon         button icon
     * @param width        preferred width
     * @param height       preferred height
     * @param cornerRadius corner radius
     * @param font         font to use (nullable)
     * @param bg           background color
     * @param fg           foreground color
     * @return configured button
     */
    public static JButton createPrimaryIconButton(
            String text,
            Icon icon,
            int width,
            int height,
            int cornerRadius,
            java.awt.Font font,
            java.awt.Color bg,
            java.awt.Color fg
    ) {
        JButton b = new JButton(text, icon) {
            @Override protected void paintComponent(Graphics g) {
                Graphics2D g2 = (Graphics2D) g.create();
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                g2.setColor(getBackground());
                g2.fillRoundRect(0, 0, getWidth(), getHeight(), cornerRadius, cornerRadius);
                g2.dispose();
                super.paintComponent(g);
            }
        };
        b.setHorizontalTextPosition(SwingConstants.RIGHT);
        b.setVerticalTextPosition(SwingConstants.CENTER);
        b.setIconTextGap(8);
        b.setContentAreaFilled(false);
        b.setOpaque(false);
        b.setBackground(bg);
        b.setForeground(fg);
        if (font != null) b.setFont(font);
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        b.setBorder(BorderFactory.createEmptyBorder(6, 12, 6, 12));
        Dimension d = new Dimension(width, height);
        b.setPreferredSize(d);
        b.setMinimumSize(d);
        b.setMaximumSize(d);
        addDarkHoverAndPress(b, bg);
        return b;
    }

    /**
     * Creates a rounded square button (size×size) that paints its own background,
     * with an icon on top and text at the bottom.
     *
     * @param text         button text
     * @param icon         button icon
```

UiUtils

```java
     * @param bg           background color
     * @param fg           foreground color
     * @param size         square size in pixels
     * @param cornerRadius corner radius
     * @param fontSize     font size
     * @return configured button
     */
    public static JButton createPaintedRoundedIconButton(
            String text,
            Icon icon,
            Color bg,
            Color fg,
            int size,
            int cornerRadius,
            float fontSize
    ) {
        JButton b = new JButton(text, icon) {
            @Override protected void paintComponent(Graphics g) {
                Graphics2D g2 = (Graphics2D) g.create();
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                g2.setColor(getBackground());
                g2.fillRoundRect(0, 0, getWidth(), getHeight(), cornerRadius, cornerRadius);
                g2.dispose();
                super.paintComponent(g);
            }
        };
        b.setContentAreaFilled(false);
        b.setOpaque(false);
        b.setBackground(bg);
        b.setForeground(fg);
        try {
            if (taskmanagement.ui.styles.AppTheme.CTRL_BUTTON_FONT != null) {
                b.setFont(taskmanagement.ui.styles.AppTheme.CTRL_BUTTON_FONT.deriveFont(fontSize));
            } else {
                b.setFont(b.getFont().deriveFont(Font.PLAIN, fontSize));
            }
        } catch (Throwable ignore) {
            b.setFont(b.getFont().deriveFont(Font.PLAIN, fontSize));
        }
        b.setHorizontalTextPosition(SwingConstants.CENTER);
        b.setVerticalTextPosition(SwingConstants.BOTTOM);
        Dimension d = new Dimension(size, size);
        b.setPreferredSize(d);
        b.setMinimumSize(d);
        b.setMaximumSize(d);
        addDarkHoverAndPress(b, bg);
        return b;
    }

    /**
     * Styles a compact rounded header button (pill-like) with stable hover/press.
     * The background is painted locally to avoid look-and-feel artifacts.
     *
     * @param b  button to style
     * @param bg background color
     * @param fg foreground (text/icon) color
     */
    public static void styleHeaderPillButton(JButton b, Color bg, Color fg) {
        b.setFocusPainted(false);
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\util\UiUtils.java

UiUtils

```java
        b.setRolloverEnabled(true);
        b.setContentAreaFilled(false);
        b.setOpaque(false);
        b.setForeground(fg);
        if (AppTheme.HB_BTN_FONT != null) b.setFont(AppTheme.HB_BTN_FONT);
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        b.setHorizontalTextPosition(SwingConstants.RIGHT);
        b.setIconTextGap(8);
        Dimension d = new Dimension(AppTheme.HB_BTN_MIN_W, AppTheme.HB_BTN_HEIGHT);
        b.setPreferredSize(d);
        b.setMinimumSize(d);
        b.setBorder(BorderFactory.createEmptyBorder(6, 12, 6, 12));
        final Color base = bg;
        final Color hover = shiftForContrast(base, 0.12f);
        final Color press = shiftForContrast(base, 0.22f);
        b.setBackground(base);
        b.getModel().addChangeListener(e -> {
            ButtonModel m = b.getModel();
            if (m.isPressed())      b.setBackground(press);
            else if (m.isRollover()) b.setBackground(hover);
            else                     b.setBackground(base);
        });
        b.setUI(new javax.swing.plaf.basic.BasicButtonUI() {
            @Override public void paint(Graphics g, JComponent c) {
                Graphics2D g2 = (Graphics2D) g.create();
                g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
                g2.setColor(b.getBackground());
                g2.fillRoundRect(0, 0, c.getWidth(), c.getHeight(), AppTheme.HB_BTN_RADIUS, AppTheme.HB_BTN_RADIUS);
                g2.dispose();
                super.paint(g, c);
            }
        });
    }
}
```

ViewModelException

```java
package taskmanagement.ui;

/**
 * Exception type for errors that occur in the ViewModel layer.
 * <p>
 * This project-specific checked exception wraps lower-level domain
 * or persistence exceptions so that the UI layer does not directly
 * depend on them.
 * </p>
 */
public class ViewModelException extends Exception {

    /**
     * Constructs a new {@code ViewModelException} with the specified detail message.
     *
     * @param message the detail message, saved for later retrieval by {@link #getMessage()}
     */
    public ViewModelException(String message) {
        super(message);
    }

    /**
     * Constructs a new {@code ViewModelException} with the specified detail message
     * and cause.
     *
     * @param message the detail message
     * @param cause   the cause (which is saved for later retrieval by {@link #getCause});
     *                a {@code null} value is permitted
     */
    public ViewModelException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\views\ContentArea.java

ContentArea

```java
package taskmanagement.ui.views;

import taskmanagement.ui.api.TasksViewAPI;
import taskmanagement.ui.widgets.ControlPanel;
import taskmanagement.ui.widgets.TasksPanel;
import taskmanagement.ui.widgets.ToolBox;

import javax.swing.*;
import java.awt.*;
import java.util.List;
import java.util.Objects;

/**
 * Three-column container composed of LEFT, CENTER, and RIGHT panes.
 * LEFT and RIGHT have fixed widths while CENTER flexes to occupy remaining space.
 * <p>
 * This view wires its child widgets to a {@link TasksViewAPI} instance and bridges
 * the current selection from {@link TasksPanel} into {@link ToolBox} so actions
 * (e.g., Advance/Mark-as) operate on the selected task IDs.
 * </p>
 */
public final class ContentArea extends JPanel {

    /** Fixed width in pixels for the left rail. */
    public static final int LEFT_PANEL_WIDTH = 150;
    /** Fixed width in pixels for the right rail. */
    public static final int RIGHT_PANEL_WIDTH = 260;
    /** Horizontal gutter between columns, in pixels. */
    public static final int H_GUTTER = 12;
    /** Minimal width for the center content, in pixels. */
    public static final int CENTER_MIN_WIDTH = 650;

    private final ControlPanel leftPanel;
    private final TasksPanel tasksPanel;
    private final ToolBox rightPanel;

    private TasksViewAPI api;

    /**
     * Creates a {@code ContentArea} with default column components.
     */
    public ContentArea() {
        this(new ControlPanel(), new TasksPanel(), new ToolBox());
    }

    /**
     * Creates a {@code ContentArea} with custom column components.
     *
     * @param leftPanel  the left rail widget; if {@code null}, a default instance is used
     * @param tasksPanel the center tasks widget; if {@code null}, a default instance is used
     * @param rightPanel the right rail widget; if {@code null}, a default instance is used
     */
    public ContentArea(ControlPanel leftPanel, TasksPanel tasksPanel, ToolBox rightPanel) {
        super(new GridBagLayout());
        this.leftPanel = (leftPanel != null) ? leftPanel : new ControlPanel();
        this.tasksPanel = (tasksPanel != null) ? tasksPanel : new TasksPanel();
        this.rightPanel = (rightPanel != null) ? rightPanel : new ToolBox();
        initLayout();
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\views\ContentArea.java

ContentArea

```java
    /**
     * Injects the {@link TasksViewAPI} and wires child widgets and interactions.
     * The right panel receives a provider for the currently selected task IDs.
     *
     * @param api the UI-facing ViewModel API
     * @throws NullPointerException if {@code api} is {@code null}
     */
    public void setApi(TasksViewAPI api) {
        this.api = Objects.requireNonNull(api, "api");
        this.leftPanel.setApi(this.api, this.tasksPanel);
        this.tasksPanel.setApi(this.api);
        this.rightPanel.setApi(this.api);
        this.rightPanel.setIdsProvider(tasksPanel::selectedIds);
        this.rightPanel.bindSelectionProperty(tasksPanel.selectedIdsProperty());
        this.rightPanel.bindTotalsFromApi();
        this.rightPanel.bindAdvanceAndMarkDialogs();
        this.rightPanel.bindSortControls(List.of(
                new taskmanagement.application.viewmodel.sort.SortById(),
                new taskmanagement.application.viewmodel.sort.SortByTitle(),
                new taskmanagement.application.viewmodel.sort.SortByState()
        ));
        this.rightPanel.bindFilterControls(this.api);
        this.api.reload();
    }

    private void initLayout() {
        setOpaque(false);
        lockWidth(leftPanel, LEFT_PANEL_WIDTH);
        lockWidth(rightPanel, RIGHT_PANEL_WIDTH);
        leftPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, H_GUTTER));
        tasksPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, H_GUTTER));
        rightPanel.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
        GridBagConstraints gc = new GridBagConstraints();
        gc.gridy = 0;
        gc.fill = GridBagConstraints.BOTH;
        gc.weighty = 1.0;
        gc.gridx = 0;
        gc.weightx = 0.0;
        add(wrap(leftPanel), gc);
        gc.gridx = 1;
        gc.weightx = 1.0;
        JComponent centerWrap = wrap(tasksPanel);
        centerWrap.setMinimumSize(new Dimension(CENTER_MIN_WIDTH, 10));
        add(centerWrap, gc);
        gc.gridx = 2;
        gc.weightx = 0.0;
        add(wrap(rightPanel), gc);
    }

    private static JComponent wrap(JComponent inner) {
        JPanel wrapper = new JPanel(new BorderLayout());
        wrapper.setOpaque(false);
        wrapper.add(inner, BorderLayout.CENTER);
        return wrapper;
    }

    private static void lockWidth(JComponent comp, int width) {
        Dimension pref = comp.getPreferredSize();
```

ContentArea

```java
        if (pref == null) pref = new Dimension(width, 10);
        Dimension fixed = new Dimension(width, Math.max(10, pref.height));
        comp.setPreferredSize(fixed);
        comp.setMinimumSize(fixed);
        comp.setMaximumSize(new Dimension(width, Integer.MAX_VALUE));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ControlPanel.java

ControlPanel

```java
package taskmanagement.ui.widgets;

import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.ui.api.TasksViewAPI;
import taskmanagement.ui.dialogs.TaskEditorDialog;
import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.util.UiUtils;

import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.List;
import java.util.Objects;
import java.util.Optional;

/**
 * Left-side control rail presenting five large square actions: Refresh, Add, Edit, Delete, and Disk Cleanup.
 * <p>
 * Icons are loaded safely with a transparent fallback. Width is fixed to prevent the center content
 * from shrinking the rail. Behavior delegates to a {@link TasksViewAPI} instance injected via
 * {@link #setApi(TasksViewAPI, TasksPanel)}.
 * </p>
 */
public final class ControlPanel extends RoundedPanel {

    private static final int INNER_PAD = 8;
    private static final int FALLBACK_WIDTH = 220;

    private final JButton refreshBtn;
    private final JButton addBtn;
    private final JButton editBtn;
    private final JButton deleteBtn;
    private final JButton cleanupBtn;

    private TasksViewAPI api;
    private TasksPanel tasksPanel;

    /**
     * Constructs the control rail with five vertically stacked buttons.
     */
    public ControlPanel() {
        super(new Color(0x2C2C2C), 16);
        setOpaque(false);

        int block = AppTheme.CTRL_BLOCK_SIZE;
        int fixedWidth = Math.max(FALLBACK_WIDTH, block + INNER_PAD * 2);
        Dimension fixed = new Dimension(fixedWidth, 10);
        setPreferredSize(fixed);
        setMinimumSize(fixed);

        setBorder(BorderFactory.createEmptyBorder(INNER_PAD, INNER_PAD, INNER_PAD, INNER_PAD));

        setLayout(new GridBagLayout());
        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.fill = GridBagConstraints.NONE;
        gbc.anchor = GridBagConstraints.CENTER;
```

ControlPanel

```java
        gbc.weightx = 1.0;

        Icon refreshIcon = safeIcon(UiUtils.loadRasterIcon("/taskmanagement/ui/resources/refresh.png",
                AppTheme.CTRL_ICON_SIZE, AppTheme.CTRL_ICON_SIZE));
        Icon addIcon = safeIcon(UiUtils.loadRasterIcon("/taskmanagement/ui/resources/add.png",
                AppTheme.CTRL_ICON_SIZE, AppTheme.CTRL_ICON_SIZE));
        Icon editIcon = safeIcon(UiUtils.loadRasterIcon("/taskmanagement/ui/resources/edit.png",
                AppTheme.CTRL_ICON_SIZE, AppTheme.CTRL_ICON_SIZE));
        Icon delIcon = safeIcon(UiUtils.loadRasterIcon("/taskmanagement/ui/resources/delete.png",
                AppTheme.CTRL_ICON_SIZE, AppTheme.CTRL_ICON_SIZE));
        Icon cleanupIcon = safeIcon(UiUtils.loadRasterIcon("/taskmanagement/ui/resources/cleanup.png",
                AppTheme.CTRL_ICON_SIZE, AppTheme.CTRL_ICON_SIZE));

        refreshBtn = UiUtils.createPaintedRoundedIconButton("Refresh", refreshIcon,
                AppTheme.CTRL_REFRESH_BG, AppTheme.CTRL_REFRESH_FG,
                block, AppTheme.CTRL_CORNER_RAD, AppTheme.CTRL_FONT_SIZE);
        gbc.gridy = 0; gbc.weighty = 1.0; add(refreshBtn, gbc);

        addBtn = UiUtils.createPaintedRoundedIconButton("Add", addIcon,
                AppTheme.CTRL_ADD_BG, AppTheme.CTRL_ADD_FG,
                block, AppTheme.CTRL_CORNER_RAD, AppTheme.CTRL_FONT_SIZE);
        gbc.gridy = 1; add(addBtn, gbc);

        editBtn = UiUtils.createPaintedRoundedIconButton("Edit", editIcon,
                AppTheme.CTRL_EDIT_BG, AppTheme.CTRL_EDIT_FG,
                block, AppTheme.CTRL_CORNER_RAD, AppTheme.CTRL_FONT_SIZE);
        gbc.gridy = 2; add(editBtn, gbc);

        deleteBtn = UiUtils.createPaintedRoundedIconButton("Delete", delIcon,
                AppTheme.CTRL_DELETE_BG, AppTheme.CTRL_DELETE_FG,
                block, AppTheme.CTRL_CORNER_RAD, AppTheme.CTRL_FONT_SIZE);
        gbc.gridy = 3; add(deleteBtn, gbc);

        cleanupBtn = UiUtils.createPaintedRoundedIconButton("Cleanup", cleanupIcon,
                AppTheme.CTRL_CLEANUP_BG, AppTheme.CTRL_CLEANUP_FG,
                block, AppTheme.CTRL_CORNER_RAD, AppTheme.CTRL_FONT_SIZE);
        gbc.gridy = 4; add(cleanupBtn, gbc);

        wireActions();
    }

    /**
     * Injects the API and the tasks panel used for selection-dependent operations.
     *
     * @param api        the {@link TasksViewAPI} implementation
     * @param tasksPanel the tasks panel providing selection information
     * @throws NullPointerException if any argument is {@code null}
     */
    public void setApi(TasksViewAPI api, TasksPanel tasksPanel) {
        this.api = Objects.requireNonNull(api, "api");
        this.tasksPanel = Objects.requireNonNull(tasksPanel, "tasksPanel");
    }

    private void wireActions() {
        refreshBtn.addActionListener(this::onRefresh);
        addBtn.addActionListener(this::onAdd);
        editBtn.addActionListener(this::onEdit);
        deleteBtn.addActionListener(this::onDelete);
        cleanupBtn.addActionListener(this::onCleanupAll);
```

ControlPanel

```java
    }

    private void onRefresh(ActionEvent e) {
        if (api != null) {
            api.reload();
        }
    }

    private void onAdd(ActionEvent e) {
        Optional<TaskEditorDialog.EditorResult> result =
                TaskEditorDialog.showDialog(this, TaskEditorDialog.Mode.ADD, null);
        result.ifPresent(r -> {
            if (api != null) {
                api.addTask(new ITask() {
                    @Override public int getId() { return 0; }
                    @Override public String getTitle() { return r.title(); }
                    @Override public String getDescription() { return r.description(); }
                    @Override public TaskState getState() { return r.state(); }
                    @Override public void accept(taskmanagement.domain.visitor.TaskVisitor v) {}
                });
            }
        });
    }

    private void onEdit(ActionEvent e) {
        if (tasksPanel == null || api == null) {
            return;
        }
        List<Integer> ids = tasksPanel.getSelectedIds();
        if (ids.size() != 1) {
            JOptionPane.showMessageDialog(this,
                    "Please select exactly one task to edit.",
                    "Edit Task",
                    JOptionPane.WARNING_MESSAGE);
            return;
        }
        int id = ids.get(0);

        api.findRowById(id).ifPresent(row -> {
            TaskEditorDialog.Prefill prefill = new TaskEditorDialog.Prefill(
                    row.id(),
                    row.title(),
                    row.description(),
                    TaskState.valueOf(row.state())
            );

            Optional<TaskEditorDialog.EditorResult> result =
                    TaskEditorDialog.showDialog(this, TaskEditorDialog.Mode.EDIT, prefill);

            result.ifPresent(r -> {
                api.updateTask(new ITask() {
                    @Override public int getId() { return r.id(); }
                    @Override public String getTitle() { return r.title(); }
                    @Override public String getDescription() { return r.description(); }
                    @Override public TaskState getState() { return r.state(); }
                    @Override public void accept(taskmanagement.domain.visitor.TaskVisitor v) {}
                });
                api.reload();
            });
```

ControlPanel

```java
        });
    }

    private void onDelete(ActionEvent e) {
        if (api != null && tasksPanel != null && showConfirmDeleteSelected()) {
            List<Integer> ids = tasksPanel.getSelectedIds();
            if (!ids.isEmpty()) {
                api.deleteTasks(ids.stream().mapToInt(Integer::intValue).toArray());
            }
        }
    }

    private void onCleanupAll(ActionEvent e) {
        if (api != null && showConfirmDeleteAll()) {
            api.deleteAll();
        }
    }

    private boolean showConfirmDeleteSelected() {
        int ans = JOptionPane.showConfirmDialog(
                this,
                "Delete selected task(s)?",
                "Confirm Delete",
                JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.WARNING_MESSAGE
        );
        return ans == JOptionPane.OK_OPTION;
    }

    private boolean showConfirmDeleteAll() {
        int ans = JOptionPane.showConfirmDialog(
                this,
                "Delete ALL tasks? This cannot be undone.",
                "Disk Cleanup",
                JOptionPane.YES_NO_OPTION,
                JOptionPane.WARNING_MESSAGE
        );
        return ans == JOptionPane.YES_OPTION;
    }

    private static Icon safeIcon(Icon icon) {
        if (icon != null) return icon;
        return new Icon() {
            @Override public void paintIcon(Component c, Graphics g, int x, int y) {}
            @Override public int getIconWidth() { return 1; }
            @Override public int getIconHeight() { return 1; }
        };
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\HeaderBar.java

HeaderBar

```java
package taskmanagement.ui.widgets;

import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.UiUtils;

import javax.swing.*;
import java.awt.*;
import java.util.function.Consumer;

/**
 * Header bar component with a left-aligned application icon and title, and right-aligned
 * pill buttons (About / Close). Colors, spacing, and typography are sourced from
 * {@link AppTheme}, and rounded buttons are styled via {@link UiUtils#styleHeaderPillButton(JButton, Color, Color)}.
 */
public class HeaderBar extends JPanel {

    private final JLabel  titleLabel;
    private final JButton aboutButton;
    private final JButton closeButton;

    private Consumer<JButton> aboutHandler;
    private Consumer<JButton> closeHandler;

    /**
     * Constructs a header bar with icon/title on the left and About/Close buttons on the right.
     */
    public HeaderBar() {
        setOpaque(true);
        setBackground(AppTheme.HEADER_BG);
        setLayout(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridy = 0;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.insets = new Insets(AppTheme.HEADER_HPAD, AppTheme.HEADER_HPAD, AppTheme.HEADER_HPAD, AppTheme.HEADER_HPAD);

        JPanel leftPanel = new JPanel(new FlowLayout(FlowLayout.LEFT, 8, 0));
        leftPanel.setOpaque(false);

        Icon appIcon = UiUtils.loadRasterIcon("/taskmanagement/ui/resources/tasks_mng.png", 40, 40);
        if (appIcon != null) {
            JLabel iconLabel = new JLabel(appIcon);
            leftPanel.add(iconLabel);
        }

        titleLabel = new JLabel("Task Management App");
        titleLabel.setForeground(AppTheme.MAIN_APP_TITLE);
        titleLabel.setFont(new Font("Segoe UI", Font.BOLD, AppTheme.TITLE_FONT));
        titleLabel.setHorizontalAlignment(SwingConstants.LEFT);
        leftPanel.add(titleLabel);

        gbc.gridx = 0;
        gbc.weightx = 0.60;
        add(leftPanel, gbc);

        JPanel actions = new JPanel(new FlowLayout(FlowLayout.RIGHT, AppTheme.ACTIONS_HGAP, AppTheme.ACTIONS_VGAP));
        actions.setOpaque(false);

        Icon infoIcon  = UiUtils.loadRasterIcon("/taskmanagement/ui/resources/information.png", 40, 40);
```

HeaderBar

```java
        Icon closeIcon = UiUtils.loadRasterIcon("/taskmanagement/ui/resources/exit.png", 50, 50);

        aboutButton = new JButton("", infoIcon);
        UiUtils.styleHeaderPillButton(aboutButton, AppTheme.HB_ABOUT_BG, AppTheme.HB_ABOUT_FG);
        aboutButton.addActionListener(e -> {
            if (aboutHandler != null) aboutHandler.accept(aboutButton);
        });

        closeButton = new JButton("", closeIcon);
        UiUtils.styleHeaderPillButton(closeButton, AppTheme.HB_CLOSE_BG, AppTheme.HB_CLOSE_FG);
        closeButton.addActionListener(e -> {
            if (closeHandler != null) closeHandler.accept(closeButton);
        });

        actions.add(aboutButton);
        actions.add(closeButton);

        gbc.gridx = 1;
        gbc.weightx = 0.40;
        add(actions, gbc);
    }

    /**
     * Sets the title text displayed on the left side of the header.
     *
     * @param text the new title text; {@code null} is treated as an empty string
     */
    public void setTitleText(String text) {
        titleLabel.setText(text != null ? text : "");
    }

    /**
     * Registers a click handler for the About button, replacing any existing listeners.
     *
     * @param handler a consumer that receives the About {@link JButton}; ignored if {@code null}
     */
    public void onAbout(Consumer<JButton> handler) {
        if (handler == null) return;
        for (var l : aboutButton.getActionListeners()) {
            aboutButton.removeActionListener(l);
        }
        aboutButton.addActionListener(e -> handler.accept(aboutButton));
        this.aboutHandler = handler;
    }

    /**
     * Registers a click handler for the Close button, replacing any existing listeners.
     *
     * @param handler a consumer that receives the Close {@link JButton}; ignored if {@code null}
     */
    public void onClose(Consumer<JButton> handler) {
        if (handler == null) return;
        for (var l : closeButton.getActionListeners()) {
            closeButton.removeActionListener(l);
        }
        closeButton.addActionListener(e -> handler.accept(closeButton));
        this.closeHandler = handler;
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\TasksPanel.java

TasksPanel

```java
package taskmanagement.ui.widgets;

import taskmanagement.application.viewmodel.events.Property; // strong-typed listener
import taskmanagement.domain.ITask;
import taskmanagement.ui.api.TasksViewAPI;
import taskmanagement.ui.util.RoundedPanel;
import taskmanagement.ui.dialogs.TaskDetailsDialog;

import javax.swing.*;
import javax.swing.border.EmptyBorder;
import java.awt.*;
import java.awt.event.ComponentAdapter;
import java.awt.event.ComponentEvent;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Objects;

/**
 * Scrollable list of tasks with a sticky header, bound to a {@link TasksViewAPI}.
 * <p>
 * The panel exposes selection via an observable property and a compatibility getter.
 * It listens to both the full tasks list and the filtered tasks list to avoid missed updates.
 * </p>
 */
public final class TasksPanel extends JPanel {

    private static final int TITLE_PREVIEW_MAX = 24;

    private static final float FONT_SIZE_BASE   = 11f;
    private static final float HEADER_FONT_SIZE = 10f;
    private static final float FONT_SIZE_PILL   = 11f;
    private static final float FONT_SIZE_ID     = 11f;

    private static final Color CARD_BG        = new Color(58, 58, 58);
    private static final Color MINI_BG        = new Color(66, 66, 66);
    private static final int   OUTER_RADIUS   = 10;
    private static final int   MINI_RADIUS    = 8;
    private static final Insets OUTER_INSETS  = new Insets(6, 8, 6, 8);
    private static final int   ROW_V_GAP      = 8;

    private static final Color HEADER_BG      = new Color(24, 24, 24);
    private static final Insets HEADER_INSETS = new Insets(6, 8, 6, 8);

    private static final Color TODO_BG   = new Color(0xE74C3C);
    private static final Color TODO_FG   = Color.WHITE;
    private static final Color INPROG_BG = Color.WHITE;
    private static final Color INPROG_FG = new Color(0x1E1E1E);
    private static final Color DONE_BG   = new Color(0x154F2A);
    private static final Color DONE_FG   = Color.WHITE;

    private static final Color PIPE_FG   = new Color(140, 140, 140);

    private static final double W_CHECK = 0.30;
    private static final double W_ID    = 0.50;
    private static final double W_TITLE = 1.40;
    private static final double W_STAT  = 1.00;
    private static final double W_BTN   = 0.90;
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\TasksPanel.java

TasksPanel

```java
    private static final double[] COL_WEIGHTS = new double[] {
            W_CHECK, 0.0,
            W_ID,    0.0,
            W_TITLE, 0.0,
            W_STAT,  0.0,
            W_BTN
    };

    private static final double[] MID_WEIGHTS = new double[] {
            0.0, W_ID, 0.0, W_TITLE, 0.0, W_STAT, 0.0
    };

    private static final int PIPE_COL_W = 12;
    private static final int MIN_CHECK_W = 30;
    private static final int MIN_ID_W    = 36;
    private static final int MIN_TITLE_W = 90;
    private static final int MIN_STAT_W  = 100;
    private static final int MIN_BTN_W   = 92;

    private final JPanel headerWrapper;
    private final RoundedPanel headerPanel;
    private final JPanel listPanel;
    private final JScrollPane scrollPane;
    private final JPanel headerRightSpacer;

    private int[] headerColsPx = null;

    private TasksViewAPI api;

    private Property.Listener<List<ITask>> tasksListener;
    private Property.Listener<List<ITask>> filteredListener;

    /** Observable property containing the currently selected task IDs (never {@code null}). */
    private final Property<int[]> selectedIdsProp = new Property<>(new int[0]);

    /**
     * Creates a new {@code TasksPanel} with a sticky header and scrollable rows.
     */
    public TasksPanel() {
        super(new BorderLayout());
        setOpaque(false);
        ensureSafeFont(this);

        headerPanel = buildHeader();
        headerWrapper = new JPanel(new BorderLayout());
        headerWrapper.setOpaque(false);
        headerWrapper.setBorder(new EmptyBorder(0, 8, ROW_V_GAP, 8));
        headerWrapper.add(headerPanel, BorderLayout.CENTER);

        headerRightSpacer = new JPanel();
        headerRightSpacer.setOpaque(false);
        headerRightSpacer.setPreferredSize(new Dimension(0, 1));
        headerWrapper.add(headerRightSpacer, BorderLayout.EAST);
        add(headerWrapper, BorderLayout.NORTH);

        listPanel = new JPanel();
        listPanel.setLayout(new BoxLayout(listPanel, BoxLayout.Y_AXIS));
        listPanel.setOpaque(false);
        listPanel.setBorder(new EmptyBorder(0, 8, 0, 8));
```

TasksPanel

```java
        ensureSafeFont(listPanel);

        scrollPane = new JScrollPane(new ScrollableWidthPanel(listPanel),
                ScrollPaneConstants.VERTICAL_SCROLLBAR_AS_NEEDED,
                ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
        scrollPane.setBorder(null);
        scrollPane.getViewport().setOpaque(false);
        scrollPane.setOpaque(false);
        ensureSafeFont(scrollPane);
        add(scrollPane, BorderLayout.CENTER);

        headerWrapper.addComponentListener(new ComponentAdapter() {
            @Override public void componentResized(ComponentEvent e) {
                SwingUtilities.invokeLater(TasksPanel.this::recomputeAndApplyFromHeader);
            }
        });
    }

    /**
     * Injects the {@link TasksViewAPI}, renders the current snapshot, and subscribes to
     * both the all-tasks and filtered-tasks properties.
     *
     * @param api the API to bind to
     * @throws NullPointerException if {@code api} is {@code null}
     */
    public void setApi(TasksViewAPI api) {
        Objects.requireNonNull(api, "api");

        if (this.api != null) {
            try {
                if (tasksListener != null) {
                    this.api.tasksProperty().removeListener(tasksListener);
                }
                if (filteredListener != null) {
                    this.api.filteredTasksProperty().removeListener(filteredListener);
                }
            } catch (Exception ignore) {
                // intentionally ignored: previous API may have been partially wired
            }
        }

        this.api = api;

        if (SwingUtilities.isEventDispatchThread()) {
            renderFromApi();
        } else {
            SwingUtilities.invokeLater(this::renderFromApi);
        }

        tasksListener = (oldList, newList) -> {
            if (SwingUtilities.isEventDispatchThread()) renderFromApi();
            else SwingUtilities.invokeLater(this::renderFromApi);
        };
        this.api.tasksProperty().addListener(tasksListener);

        filteredListener = (oldList, newList) -> {
            if (SwingUtilities.isEventDispatchThread()) renderFromApi();
            else SwingUtilities.invokeLater(this::renderFromApi);
        };
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\TasksPanel.java

TasksPanel

```java
        this.api.filteredTasksProperty().addListener(filteredListener);
    }

    /**
     * Forces a refresh using the latest snapshot from the bound API.
     */
    public void refreshNow() {
        if (SwingUtilities.isEventDispatchThread()) {
            renderFromApi();
        } else {
            SwingUtilities.invokeLater(this::renderFromApi);
        }
    }

    /**
     * Returns a live observable property of the currently selected task IDs.
     *
     * @return the selection property, never {@code null}
     */
    public Property<int[]> selectedIdsProperty() {
        return selectedIdsProp;
    }

    /**
     * Returns the currently selected task IDs as a defensive copy.
     *
     * @return an array of selected IDs (never {@code null})
     */
    public int[] selectedIds() {
        int[] v = selectedIdsProp.getValue();
        return (v == null) ? new int[0] : v.clone();
    }

    /**
     * Returns the selected task IDs as a list (compatibility API).
     *
     * @return list of selected IDs, never {@code null}
     */
    public List<Integer> getSelectedIds() {
        List<Integer> ids = new ArrayList<>();
        for (Component c : listPanel.getComponents()) {
            if (c instanceof RoundedPanel row) {
                JCheckBox chk = (JCheckBox) findByName(row, "row-check");
                if (chk != null && chk.isSelected()) {
                    Integer id = (Integer) chk.getClientProperty("taskId");
                    if (id != null) ids.add(id);
                }
            }
        }
        return ids;
    }

    private void renderFromApi() {
        if (api == null) return;
        List<ITask> filtered = api.filteredTasksProperty().getValue();
        List<ITask> all      = api.tasksProperty().getValue();
        List<ITask> toShow   = (filtered != null) ? filtered : all;
        renderTasks(toShow);
    }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\TasksPanel.java

TasksPanel

```java
    private void renderTasks(List<ITask> tasks) {
        int[] oldSel = selectedIds();

        listPanel.removeAll();
        if (tasks != null) {
            int n = tasks.size();
            for (int i = 0; i < n; i++) {
                ITask t = tasks.get(i);
                listPanel.add(buildRow(t));
                if (i < n - 1) {
                    listPanel.add(Box.createVerticalStrut(ROW_V_GAP));
                }
            }
        }
        listPanel.revalidate();
        listPanel.repaint();

        if (oldSel.length > 0) {
            for (Component c : listPanel.getComponents()) {
                if (c instanceof RoundedPanel row) {
                    JCheckBox chk = (JCheckBox) findByName(row, "row-check");
                    if (chk != null) {
                        Integer id = (Integer) chk.getClientProperty("taskId");
                        if (id != null && contains(oldSel, id)) chk.setSelected(true);
                    }
                }
            }
        }
        fireSelectionChanged();

        SwingUtilities.invokeLater(this::recomputeAndApplyFromHeader);
    }

    private RoundedPanel buildHeader() {
        RoundedPanel header = new RoundedPanel(HEADER_BG, 10);
        header.setOpaque(false);
        header.setLayout(new GridBagLayout());
        header.setBorder(new EmptyBorder(HEADER_INSETS));
        ensureSafeFont(header);

        ((GridBagLayout) header.getLayout()).columnWeights = COL_WEIGHTS.clone();

        GridBagConstraints gc = new GridBagConstraints();
        gc.gridy = 0;
        gc.insets = new Insets(0, 0, 0, 6);
        gc.anchor = GridBagConstraints.CENTER;
        gc.fill   = GridBagConstraints.HORIZONTAL;
        gc.weighty = 0;

        int x = 0;
        headerAdd(header, gc, x++, headerLabel("▯"),      W_CHECK);
        headerAdd(header, gc, x++, pipeLabel(),           0);
        headerAdd(header, gc, x++, headerLabel("ID"),     W_ID);
        headerAdd(header, gc, x++, pipeLabel(),           0);
        headerAdd(header, gc, x++, headerLabel("Title"),  W_TITLE);
        headerAdd(header, gc, x++, pipeLabel(),           0);
        headerAdd(header, gc, x++, headerLabel("Status"), W_STAT);
        headerAdd(header, gc, x++, pipeLabel(),           0);
```

TasksPanel

```java
        headerAdd(header, gc, x++, headerLabel("[...]"),  W_BTN);

        header.setAlignmentX(LEFT_ALIGNMENT);
        header.setMaximumSize(new Dimension(Integer.MAX_VALUE, header.getPreferredSize().height));
        return header;
    }

    private static void headerAdd(JPanel header, GridBagConstraints gc, int gridx, JComponent comp, double weightx) {
        GridBagConstraints c = (GridBagConstraints) gc.clone();
        c.gridx = gridx; c.weightx = weightx;
        header.add(comp, c);
    }

    private static JLabel headerLabel(String text) {
        JLabel l = new JLabel(text, SwingConstants.CENTER);
        ensureSafeFont(l);
        l.setForeground(Color.WHITE);
        l.setFont(safeDerive(l.getFont(), Font.BOLD).deriveFont(HEADER_FONT_SIZE));
        return l;
    }

    private static JLabel pipeLabel() {
        JLabel p = new JLabel("|", SwingConstants.CENTER);
        ensureSafeFont(p);
        p.setForeground(PIPE_FG);
        p.setFont(p.getFont().deriveFont(FONT_SIZE_BASE));
        return p;
    }

    private RoundedPanel buildRow(ITask task) {
        RoundedPanel row = new RoundedPanel(CARD_BG, OUTER_RADIUS);
        row.setOpaque(false);
        row.setBorder(new EmptyBorder(OUTER_INSETS));
        row.setLayout(new GridBagLayout());
        ensureSafeFont(row);

        GridBagConstraints gc = new GridBagConstraints();
        gc.gridy = 0;
        gc.insets = new Insets(0, 0, 0, 6);
        gc.anchor = GridBagConstraints.CENTER;
        gc.fill   = GridBagConstraints.BOTH;

        RoundedPanel left = new RoundedPanel(MINI_BG, MINI_RADIUS);
        left.setOpaque(false);
        left.setLayout(new GridBagLayout());
        left.setBorder(new EmptyBorder(4, 6, 4, 6));
        JCheckBox chk = new JCheckBox();
        chk.setName("row-check");
        chk.putClientProperty("taskId", task.getId());
        chk.setOpaque(false);
        chk.setFocusable(false);
        Dimension cbSize = new Dimension(18, 18);
        chk.setPreferredSize(cbSize);
        chk.setMinimumSize(cbSize);
        chk.addActionListener(e -> fireSelectionChanged());
        left.add(chk, new GridBagConstraints());
        gc.gridx = 0; gc.weightx = 0;
        row.add(left, gc);
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\TasksPanel.java

TasksPanel

```java
        RoundedPanel mid = new RoundedPanel(MINI_BG, MINI_RADIUS);
        mid.setOpaque(false);
        mid.setLayout(new GridBagLayout());
        mid.setBorder(new EmptyBorder(4, 6, 4, 6));
        mid.putClientProperty("role", "mid-panel");

        GridBagConstraints g = new GridBagConstraints();
        g.gridy = 0;
        g.insets = new Insets(0, 0, 0, 6);
        g.anchor = GridBagConstraints.CENTER;
        g.fill   = GridBagConstraints.HORIZONTAL;

        int xx = 0;
        addMid(mid, g, xx++, pipeLabel(), 0);

        JLabel id = centeredLabel(String.valueOf(task.getId()));
        id.setFont(safeDerive(id.getFont(), Font.BOLD).deriveFont(FONT_SIZE_ID));
        addMid(mid, g, xx++, id, W_ID);

        addMid(mid, g, xx++, pipeLabel(), 0);

        JLabel title = centeredLabel(clipForPreview(task.getTitle(), TITLE_PREVIEW_MAX));
        title.setFont(title.getFont().deriveFont(FONT_SIZE_BASE));
        addMid(mid, g, xx++, title, W_TITLE);

        addMid(mid, g, xx++, pipeLabel(), 0);

        JLabel status = pill(task.getState().name());
        status.setFont(safeDerive(status.getFont(), Font.BOLD).deriveFont(FONT_SIZE_PILL));
        addMid(mid, g, xx++, centerWrap(status), W_STAT);

        addMid(mid, g, xx++, pipeLabel(), 0);

        gc.gridx = 1; gc.weightx = 1;
        row.add(mid, gc);

        RoundedPanel right = new RoundedPanel(MINI_BG, MINI_RADIUS);
        right.setOpaque(false);
        right.setLayout(new GridBagLayout());
        right.setBorder(new EmptyBorder(4, 6, 4, 6));
        JButton more = new JButton("Show more");
        ensureSafeFont(more);
        more.setFont(more.getFont().deriveFont(FONT_SIZE_BASE));
        more.setBorder(BorderFactory.createEmptyBorder(2, 6, 2, 6));
        more.addActionListener(e -> TaskDetailsDialog.showDialog(row, task));
        right.add(more, new GridBagConstraints());
        gc.gridx = 2; gc.weightx = 0; gc.insets = new Insets(0, 0, 0, 0);
        row.add(right, gc);

        row.setAlignmentX(LEFT_ALIGNMENT);
        Dimension pref = row.getPreferredSize();
        row.setMaximumSize(new Dimension(Integer.MAX_VALUE, pref.height));

        if (headerColsPx != null) syncRowLayouts(row);
        return row;
    }

    private static void addMid(JPanel panel, GridBagConstraints base, int gridx, JComponent comp, double weightx) {
        GridBagConstraints c = (GridBagConstraints) base.clone();
```

TasksPanel

```java
        c.gridx = gridx; c.weightx = weightx;
        panel.add(comp, c);
    }

    private void recomputeAndApplyFromHeader() {
        JScrollBar vsb = scrollPane.getVerticalScrollBar();
        int sbw = (vsb != null && vsb.isShowing()) ? Math.max(0, vsb.getWidth()) : 0;
        headerRightSpacer.setPreferredSize(new Dimension(sbw, 1));
        headerRightSpacer.setMinimumSize(new Dimension(sbw, 1));
        headerRightSpacer.setMaximumSize(new Dimension(sbw, Integer.MAX_VALUE));
        headerWrapper.revalidate();

        int available = headerPanel.getWidth();
        if (available <= 0) { SwingUtilities.invokeLater(this::recomputeAndApplyFromHeader); return; }

        int inner = available - (HEADER_INSETS.left + HEADER_INSETS.right);
        if (inner <= 0) return;

        final int PIPE_COUNT = 4;
        int pipesWidth = PIPE_COUNT * PIPE_COL_W;

        int[] mins = { MIN_CHECK_W, MIN_ID_W, MIN_TITLE_W, MIN_STAT_W, MIN_BTN_W };
        double[] ws = { W_CHECK, W_ID, W_TITLE, W_STAT, W_BTN };

        int contentWidth = inner - pipesWidth;
        int minSum = Arrays.stream(mins).sum();

        int[] cols = mins.clone();
        int extra = Math.max(0, contentWidth - minSum);
        double wSum = Arrays.stream(ws).sum();
        for (int i = 0; i < cols.length; i++) {
            cols[i] += (int) Math.floor(extra * (ws[i] / wSum));
        }
        int used = Arrays.stream(cols).sum();
        int delta = contentWidth - used;
        if (delta != 0) cols[2] += delta;

        headerColsPx = new int[] {
                cols[0], PIPE_COL_W,
                cols[1], PIPE_COL_W,
                cols[2], PIPE_COL_W,
                cols[3], PIPE_COL_W,
                cols[4]
        };

        GridBagLayout h = (GridBagLayout) headerPanel.getLayout();
        h.columnWeights = COL_WEIGHTS.clone();
        h.columnWidths  = headerColsPx.clone();
        headerPanel.revalidate();

        for (Component c : listPanel.getComponents()) {
            if (c instanceof RoundedPanel row) syncRowLayouts(row);
        }
        listPanel.revalidate();
        listPanel.repaint();
    }

    private void syncRowLayouts(RoundedPanel row) {
        int leftW  = headerColsPx[0];
        int midW   = 0; for (int i = 1; i <= 7; i++) midW += headerColsPx[i];
        int rightW = headerColsPx[8];

        GridBagLayout outer = (GridBagLayout) row.getLayout();
        outer.columnWeights = new double[] { 0, 1, 0 };
        outer.columnWidths  = new int[] { leftW, midW, rightW };

        for (Component cc : row.getComponents()) {
            if (cc instanceof RoundedPanel p && "mid-panel".equals(p.getClientProperty("role"))) {
                if (p.getLayout() instanceof GridBagLayout mid) {
                    int[] midCols = Arrays.copyOfRange(headerColsPx, 1, 8);
                    mid.columnWeights = MID_WEIGHTS.clone();
                    mid.columnWidths  = midCols;
                    p.revalidate();
                }
            }
        }
        row.revalidate();
    }

    /**
     * Recomputes the current selection from visible rows and updates the selection property.
     */
    private void fireSelectionChanged() {
        List<Integer> ids = getSelectedIds();
        int[] now = ids.stream().mapToInt(Integer::intValue).toArray();
        int[] prev = selectedIdsProp.getValue();
        if (!Arrays.equals(prev, now)) {
            selectedIdsProp.setValue(now);
        }
    }

    private static boolean contains(int[] arr, int id) {
        for (int v : arr) if (v == id) return true;
        return false;
    }

    private static String clipForPreview(String s, int n) {
        if (s == null) return "";
        if (n <= 3)    return (s.length() <= 3) ? s : "...";
        if (s.length() <= n) return s;
        return s.substring(0, n - 3) + "...";
    }

    private static JComponent centerWrap(JComponent c) {
        JPanel p = new JPanel(new GridBagLayout());
        p.setOpaque(false);
        p.add(c, new GridBagConstraints());
        return p;
    }

    private static JLabel centeredLabel(String s) {
        JLabel l = new JLabel(s, SwingConstants.CENTER);
        ensureSafeFont(l);
        l.setForeground(new Color(235, 235, 235));
        l.setFont(l.getFont().deriveFont(FONT_SIZE_BASE));
        return l;
    }
```

TasksPanel

```java
    private static JLabel dim(String s) {
        JLabel l = new JLabel(s);
        ensureSafeFont(l);
        l.setForeground(new Color(200, 200, 200));
        l.setFont(safeDerive(l.getFont(), Font.BOLD).deriveFont(FONT_SIZE_BASE));
        return l;
    }

    private static JLabel val(String s) {
        JLabel l = new JLabel(s);
        ensureSafeFont(l);
        l.setForeground(new Color(235, 235, 235));
        l.setFont(l.getFont().deriveFont(FONT_SIZE_BASE));
        return l;
    }

    private static JLabel pill(String status) {
        String s = status == null ? "" : status;
        JLabel l = new JLabel(s, SwingConstants.CENTER);
        ensureSafeFont(l);
        l.setOpaque(true);
        l.setBorder(BorderFactory.createEmptyBorder(2, 6, 2, 6));

        Color bg, fg;
        switch (s.toLowerCase()) {
            case "to-do", "todo" -> { bg = TODO_BG;   fg = TODO_FG; }
            case "in-progress", "in progress", "inprog" -> { bg = INPROG_BG; fg = INPROG_FG; }
            case "completed", "done" -> { bg = DONE_BG;   fg = DONE_FG; }
            default -> { bg = new Color(90, 90, 90); fg = new Color(240, 240, 240); }
        }
        l.setBackground(bg);
        l.setForeground(fg);
        l.setFont(safeDerive(l.getFont(), Font.BOLD).deriveFont(FONT_SIZE_PILL));
        return l;
    }

    private static JComponent findByName(Container parent, String name) {
        for (Component c : parent.getComponents()) {
            if (name.equals(c.getName())) return (JComponent) c;
            if (c instanceof Container nested) {
                JComponent f = findByName(nested, name);
                if (f != null) return f;
            }
        }
        return null;
    }

    private static final class ScrollableWidthPanel extends JPanel implements Scrollable {
        private final JComponent inner;
        ScrollableWidthPanel(JComponent inner) {
            super(new BorderLayout());
            this.inner = inner;
            add(inner, BorderLayout.NORTH);
            setOpaque(false);
            ensureSafeFont(this);
            ensureSafeFont(inner);
        }
        @Override public Dimension getPreferredScrollableViewportSize() { return inner.getPreferredSize(); }
        @Override public int getScrollableUnitIncrement(Rectangle r, int o, int d) { return 24; }
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\TasksPanel.java

TasksPanel

```java
        @Override public int getScrollableBlockIncrement(Rectangle r, int o, int d) { return Math.max(r.height - 24, 24); }
        @Override public boolean getScrollableTracksViewportWidth() { return true; }
        @Override public boolean getScrollableTracksViewportHeight() { return false; }
    }

    private static void ensureSafeFont(JComponent c) {
        if (c.getFont() == null) {
            Font f = UIManager.getFont("Label.font");
            if (f == null) f = new JLabel().getFont();
            c.setFont(f);
        }
    }

    private static Font safeDerive(Font base, int style) {
        if (base == null) {
            Font f = UIManager.getFont("Label.font");
            if (f == null) f = new JLabel().getFont();
            return f.deriveFont(style);
        }
        return base.deriveFont(style);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ToolBox.java

ToolBox

```java
package taskmanagement.ui.widgets;

import taskmanagement.ui.styles.AppTheme;
import taskmanagement.ui.util.UiUtils;
import taskmanagement.ui.util.RoundedPanel;

import taskmanagement.domain.ITask;
import taskmanagement.application.viewmodel.events.Property;
import taskmanagement.application.viewmodel.sort.SortStrategy;

import taskmanagement.domain.TaskState;
import taskmanagement.domain.filter.ITaskFilter;
import taskmanagement.domain.filter.Filters;
import taskmanagement.ui.api.TasksViewAPI;

// ===== Export wiring =====
import taskmanagement.ui.dialogs.ExportDialog;
import taskmanagement.application.viewmodel.ExportFormat;

import javax.swing.*;
import java.awt.*;
import java.nio.file.Path;
import java.nio.file.Files; // □ added: used to verify export actually wrote a file
import java.util.EnumSet;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.util.Objects;
import java.util.Optional;
import java.util.function.Function;
import java.util.function.Supplier;
import java.util.stream.IntStream;

/**
 * A right-side toolbox composed of six sections (Undo/Redo, Advance/Mark, Sort, Filter, Counters, Export).
 * <p>
 * This view delegates user actions to {@link TasksViewAPI} (MVVM). It exposes wiring helpers to bind
 * selection, sorting strategies, filters (Combinator), and exporting behavior.
 * </p>
 */
public final class ToolBox extends RoundedPanel {

    private static final int INNER_PAD = 12;
    private static final int FALLBACK_WIDTH = 360;

    // === Section 1: Undo/Redo ===
    private final JButton undoBtn = new JButton("Undo");
    private final JButton redoBtn = new JButton("Redo");

    // === Section 2: Advance / Mark as… ===
    private final JButton advanceBtn = new JButton("Advance");
    private final JButton markAsBtn  = new JButton("Mark as…");

    // === Section 3: Sort By + Apply/Reset ===
    private final JComboBox<String> sortCombo = new JComboBox<>();
    private final JButton sortApplyBtn = new JButton("Apply");
    private final JButton sortResetBtn = new JButton("Reset");

    // === Section 4: Filter (title + checkboxes) + Apply / Reset / Show Filtered ===
```

ToolBox

```java
    private final JTextField titleField = new JTextField();
    private final JCheckBox cbTodo       = new JCheckBox("To-Do");
    private final JCheckBox cbInProgress = new JCheckBox("In-Progress");
    private final JCheckBox cbCompleted  = new JCheckBox("Completed");
    private final JButton       filterApplyBtn  = new JButton("Apply");
    private final JButton       filterResetBtn  = new JButton("Reset");
    private final JToggleButton showFilteredTgl = new JToggleButton("Count filtered as total");

    // === Section 5: Counters (Selected / Total) ===
    private final JLabel selectedCountLbl = new JLabel("Selected: 0", SwingConstants.CENTER);
    private final JLabel totalCountLbl    = new JLabel("Total: 0", SwingConstants.CENTER);

    // === Section 6: Export ===
    private JButton exportBtn;

    // ---- Binding state ----
    private TasksViewAPI api;
    private IdsProvider idsProvider;
    private Function<String, SortStrategy> sortMapper;
    private Supplier<ITaskFilter> filterSupplier;
    private ExportHandler exportHandler;
    private Property<int[]> selectionProp;

    // Keep listeners to avoid GC (when bound via properties)
    private Property.Listener<List<ITask>> tasksListener;
    private Property.Listener<int[]>       selectionListener;
    private Property.Listener<List<ITask>> filteredListener;

    // Internal name->strategy map for bindSortControls
    private final Map<String, SortStrategy> sortMap = new LinkedHashMap<>();

    /**
     * Creates the toolbox panel with all six sections and placeholder wiring.
     */
    public ToolBox() {
        super(AppTheme.PANEL_BG, AppTheme.TB_CORNER_RADIUS);
        setOpaque(false);

        int fixedWidth = Math.max(FALLBACK_WIDTH, AppTheme.TB_EXPORT_W + INNER_PAD * 2);
        Dimension fixed = new Dimension(fixedWidth, 10);
        setPreferredSize(fixed);
        setMinimumSize(fixed);

        setBorder(BorderFactory.createEmptyBorder(INNER_PAD, INNER_PAD, INNER_PAD, INNER_PAD));

        setLayout(new GridBagLayout());
        GridBagConstraints root = new GridBagConstraints();
        root.gridx = 0;
        root.fill = GridBagConstraints.BOTH;
        root.weightx = 1.0;

        // Section layout proportions.
        root.gridy = 0; root.weighty = 0.20;
        add(buildSection1_UndoRedo(), root);

        root.gridy = 1; root.weighty = 0.20;
        add(buildSection2_AdvanceMark(), root);

        root.gridy = 2; root.weighty = 0.10;
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ToolBox.java

ToolBox

```java
        add(buildSection3_Sort(), root);

        root.gridy = 3; root.weighty = 0.30;
        add(buildSection4_Filter(), root);

        root.gridy = 4; root.weighty = 0.05;
        add(buildSection5_Counters(), root);

        root.gridy = 5; root.weighty = 0.15;
        add(buildSection6_Export(), root);

        wirePlaceholders();
    }

    private JPanel buildSection1_UndoRedo() {
        JPanel p = makeTransparent();
        p.setLayout(new GridLayout(1, 2, AppTheme.TB_GAP_SM, 0));

        JPanel left  = centerInGridBag(undoBtn);
        JPanel right = centerInGridBag(redoBtn);

        styleMiniRound(undoBtn, AppTheme.TB_UNDO_BG, AppTheme.TB_UNDO_FG);
        styleMiniRound(redoBtn, AppTheme.TB_REDO_BG, AppTheme.TB_REDO_FG);

        p.add(left);
        p.add(right);
        return p;
    }

    private JPanel buildSection2_AdvanceMark() {
        JPanel p = makeTransparent();
        p.setLayout(new GridLayout(1, 2, AppTheme.TB_GAP_SM, 0));

        Icon advIcon = safeIcon(UiUtils.loadRasterIcon(
                "/taskmanagement/ui/resources/advance.png",
                AppTheme.TB_ACTION_ICON, AppTheme.TB_ACTION_ICON));
        Icon markIcon = safeIcon(UiUtils.loadRasterIcon(
                "/taskmanagement/ui/resources/mark.png",
                AppTheme.TB_ACTION_ICON, AppTheme.TB_ACTION_ICON));

        styleIconTextButton(advanceBtn, advIcon, AppTheme.TB_ADVANCE_BG, AppTheme.TB_ADVANCE_FG);
        styleIconTextButton(markAsBtn,  markIcon, AppTheme.TB_MARK_BG,    AppTheme.TB_MARK_FG);

        p.add(centerInGridBag(advanceBtn));
        p.add(centerInGridBag(markAsBtn));
        return p;
    }

    private JPanel buildSection3_Sort() {
        JPanel p = makeTransparent();
        p.setLayout(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0; gbc.weightx = 1.0; gbc.fill = GridBagConstraints.HORIZONTAL;

        JPanel top = makeTransparent();
        top.setLayout(new GridBagLayout());
        JLabel sortLbl = new JLabel("Sort by");
        sortLbl.setForeground(AppTheme.TB_TEXT_FG);
```

ToolBox

```java
        sortCombo.setPreferredSize(new Dimension(AppTheme.TB_FIELD_WIDTH, AppTheme.TB_FIELD_HEIGHT));

        GridBagConstraints t = new GridBagConstraints();
        t.insets = new Insets(AppTheme.TB_PAD, AppTheme.TB_PAD, AppTheme.TB_PAD / 2, AppTheme.TB_PAD);
        t.gridx = 0; t.gridy = 0; t.anchor = GridBagConstraints.WEST;
        top.add(sortLbl, t);
        t.gridx = 1; t.gridy = 0; t.weightx = 1.0; t.fill = GridBagConstraints.HORIZONTAL;
        top.add(sortCombo, t);

        gbc.gridy = 0; gbc.weighty = 0.5;
        p.add(top, gbc);

        JPanel bottom = makeTransparent();
        bottom.setLayout(new GridLayout(1, 2, AppTheme.TB_GAP_SM, 0));
        styleSmallFilled(sortApplyBtn, AppTheme.TB_SORT_APPLY_BG, AppTheme.TB_SORT_APPLY_FG);
        styleSmallFilled(sortResetBtn, AppTheme.TB_SORT_RESET_BG, AppTheme.TB_SORT_RESET_FG);
        bottom.add(sortApplyBtn);
        bottom.add(sortResetBtn);

        gbc.gridy = 1; gbc.weighty = 0.5;
        p.add(bottom, gbc);

        return p;
    }

    private JPanel buildSection4_Filter() {
        JPanel p = makeTransparent();
        p.setLayout(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();
        gbc.gridx = 0;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        gbc.weightx = 1.0;

        JPanel search = makeTransparent();
        search.setLayout(new GridBagLayout());
        JLabel titleLbl = new JLabel("Title contains");
        titleLbl.setForeground(AppTheme.TB_TEXT_FG);
        titleField.setPreferredSize(new Dimension(AppTheme.TB_FIELD_WIDTH, AppTheme.TB_FIELD_HEIGHT));

        GridBagConstraints s = new GridBagConstraints();
        s.insets = new Insets(AppTheme.TB_PAD, AppTheme.TB_PAD, AppTheme.TB_PAD / 2, AppTheme.TB_PAD);
        s.gridx = 0; s.gridy = 0; s.anchor = GridBagConstraints.WEST;
        search.add(titleLbl, s);
        s.gridx = 1; s.gridy = 0; s.weightx = 1.0; s.fill = GridBagConstraints.HORIZONTAL;
        search.add(titleField, s);

        gbc.gridy = 0;
        gbc.weighty = 0.15;
        p.add(search, gbc);

        JPanel list = makeTransparent();
        list.setLayout(new GridBagLayout());

        styleCheck(cbTodo);
        styleCheck(cbInProgress);
        styleCheck(cbCompleted);

        GridBagConstraints r = new GridBagConstraints();
```

ToolBox

```
        r.gridx = 0; r.weightx = 1.0; r.anchor = GridBagConstraints.WEST;
        r.insets = new Insets(2, AppTheme.TB_PAD, 2, AppTheme.TB_PAD);
        r.gridy = 0; list.add(cbTodo, r);
        r.gridy = 1; list.add(cbInProgress, r);
        r.gridy = 2; list.add(cbCompleted, r);

        gbc.gridy = 1;
        gbc.weighty = 0.55;
        gbc.fill = GridBagConstraints.HORIZONTAL;
        p.add(list, gbc);

        JPanel buttonsArea = makeTransparent();
        buttonsArea.setLayout(new GridBagLayout());

        GridBagConstraints bb = new GridBagConstraints();
        bb.gridx = 0; bb.fill = GridBagConstraints.HORIZONTAL; bb.weightx = 1.0;

        JPanel topRow = makeTransparent();
        topRow.setLayout(new GridLayout(1, 2, AppTheme.TB_GAP_SM, 0));
        styleSmallFilled(filterApplyBtn, AppTheme.TB_FILTER_APPLY_BG, AppTheme.TB_FILTER_APPLY_FG);
        styleSmallFilled(filterResetBtn, AppTheme.TB_FILTER_RESET_BG, AppTheme.TB_FILTER_RESET_FG);

        Dimension smallBtn = new Dimension(90, 28);
        filterApplyBtn.setPreferredSize(smallBtn);
        filterResetBtn.setPreferredSize(smallBtn);

        topRow.add(filterApplyBtn);
        topRow.add(filterResetBtn);

        bb.gridy = 0;
        bb.weighty = 0.5;
        buttonsArea.add(topRow, bb);

        JPanel bottomRow = makeTransparent();
        bottomRow.setLayout(new GridBagLayout());

        styleSmallHollow(
                showFilteredTgl,
                AppTheme.TB_SHOW_BORDER,
                AppTheme.TB_SHOW_FG,
                AppTheme.TB_SHOW_SELECTED_BG,
                AppTheme.TB_SHOW_SELECTED_FG
        );
        showFilteredTgl.setText("Count filtered as total \u25BE");
        showFilteredTgl.setHorizontalAlignment(SwingConstants.CENTER);

        GridBagConstraints bt = new GridBagConstraints();
        bt.gridx = 0; bt.gridy = 0;
        bt.insets = new Insets(AppTheme.TB_PAD / 2, AppTheme.TB_PAD, AppTheme.TB_PAD / 2, AppTheme.TB_PAD);
        bt.fill = GridBagConstraints.HORIZONTAL;
        bt.weightx = 1.0;
        bottomRow.add(showFilteredTgl, bt);

        bb.gridy = 1;
        bb.weighty = 0.5;
        buttonsArea.add(bottomRow, bb);

        gbc.gridy = 2;
        gbc.weighty = 0.30;
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ToolBox.java

ToolBox

```java
        gbc.fill = GridBagConstraints.BOTH;
        p.add(buttonsArea, gbc);

        return p;
    }

    private JPanel buildSection5_Counters() {
        JPanel p = makeTransparent();
        p.setLayout(new GridLayout(1, 2, AppTheme.TB_GAP_SM, 0));

        selectedCountLbl.setForeground(AppTheme.TB_TEXT_FG);
        totalCountLbl.setForeground(AppTheme.TB_TEXT_FG);

        p.add(centerInGridBag(selectedCountLbl));
        p.add(centerInGridBag(totalCountLbl));
        return p;
    }

    private JPanel buildSection6_Export() {
        JPanel p = makeTransparent();
        p.setLayout(new GridBagLayout());

        Icon exportIcon = safeIcon(UiUtils.loadRasterIcon(
                "/taskmanagement/ui/resources/download.png",
                AppTheme.TB_EXPORT_ICON, AppTheme.TB_EXPORT_ICON));

        exportBtn = UiUtils.createPrimaryIconButton(
                "Export",
                exportIcon,
                AppTheme.TB_EXPORT_W,
                AppTheme.TB_EXPORT_H,
                AppTheme.TB_EXPORT_RADIUS,
                AppTheme.TB_EXPORT_FONT,
                AppTheme.TB_EXPORT_BG,
                AppTheme.TB_EXPORT_FG
        );

        GridBagConstraints c = new GridBagConstraints();
        c.gridx = 0; c.gridy = 0; c.anchor = GridBagConstraints.CENTER;
        c.insets = new Insets(AppTheme.TB_PAD, AppTheme.TB_PAD, AppTheme.TB_PAD, AppTheme.TB_PAD);
        p.add(exportBtn, c);

        return p;
    }

    private void wirePlaceholders() {
        // Default placeholder actions until MVVM wiring is provided.
        undoBtn.addActionListener(e -> JOptionPane.showMessageDialog(this,"Undo (placeholder)","Undo",JOptionPane.INFORMATION_MESSAGE));
        redoBtn.addActionListener(e -> JOptionPane.showMessageDialog(this,"Redo (placeholder)","Redo",JOptionPane.INFORMATION_MESSAGE));
        advanceBtn.addActionListener(e -> JOptionPane.showMessageDialog(this,"Advance (placeholder)","Advance",JOptionPane.INFORMATION_MESSAGE));
        markAsBtn.addActionListener(e -> JOptionPane.showMessageDialog(this,"Mark as… (placeholder)","Mark",JOptionPane.INFORMATION_MESSAGE));

        filterApplyBtn.addActionListener(e -> JOptionPane.showMessageDialog(this,"Filter Apply (placeholder)","Filter",JOptionPane.INFORMATION_MESSAGE));
        filterResetBtn.addActionListener(e -> JOptionPane.showMessageDialog(this,"Filter Reset (placeholder)","Filter",JOptionPane.INFORMATION_MESSAGE));
        showFilteredTgl.addActionListener(e -> updateTotalsFromApi());

        if (exportBtn != null) {
            for (var l : exportBtn.getActionListeners()) exportBtn.removeActionListener(l);
            exportBtn.addActionListener(e -> openExportDialogAndRun());
```

ToolBox

```
        }
    }

    /**
     * Full binding helper (actions, counters, filters, export).
     *
     * @param api               the UI-facing API
     * @param idsProvider       provider for selected task IDs
     * @param sortMapper        mapping from combo text to a {@link SortStrategy}
     * @param filterSupplier    supplier for composed {@link ITaskFilter}
     * @param exportHandler     handler that performs export
     * @param selectionProperty observable selection property
     */
    public void wireTo(TasksViewAPI api,
                       IdsProvider idsProvider,
                       Function<String, SortStrategy> sortMapper,
                       Supplier<ITaskFilter> filterSupplier,
                       ExportHandler exportHandler,
                       Property<int[]> selectionProperty) {

        setApi(api);
        setIdsProvider(idsProvider);
        setSortMapper(sortMapper);
        setFilterSupplier(filterSupplier);
        setExportHandler(exportHandler);
        bindSelectionProperty(selectionProperty);
        bindTotalsFromApi();
        bindAdvanceAndMarkDialogs();
    }

    /**
     * Binds the API and wires safe default actions (undo/redo and filter reset).
     *
     * @param api the {@link TasksViewAPI} to use
     * @throws NullPointerException if api is null
     */
    public void setApi(TasksViewAPI api) {
        this.api = Objects.requireNonNull(api, "api");

        for (var l : undoBtn.getActionListeners()) undoBtn.removeActionListener(l);
        for (var l : redoBtn.getActionListeners()) redoBtn.removeActionListener(l);
        for (var l : filterResetBtn.getActionListeners()) filterResetBtn.removeActionListener(l);

        undoBtn.addActionListener(e -> this.api.undo());
        redoBtn.addActionListener(e -> this.api.redo());

        filterResetBtn.addActionListener(e -> {
            clearFilterUI();
            this.api.clearFilter();
            updateTotalsFromApi();
        });

        if (exportHandler == null && exportBtn != null) {
            for (var l : exportBtn.getActionListeners()) exportBtn.removeActionListener(l);
            exportBtn.addActionListener(e -> openExportDialogAndRun());
        }
    }

    /**
```

```java
     * Sets the provider used to obtain currently selected task IDs.
     *
     * @param idsProvider provider returning selected IDs
     * @throws NullPointerException if idsProvider is null
     */
    public void setIdsProvider(IdsProvider idsProvider) {
        this.idsProvider = Objects.requireNonNull(idsProvider, "idsProvider");
        updateSelectionCount();
        enableActionButtons();
    }

    /**
     * Populates the sort combo and wires Apply/Reset using the first entry as the default.
     *
     * @param strategies strategies to expose
     * @throws NullPointerException if strategies is null
     */
    public void bindSortControls(List<SortStrategy> strategies) {
        Objects.requireNonNull(strategies, "strategies");
        sortMap.clear();

        DefaultComboBoxModel<String> model = new DefaultComboBoxModel<>();
        for (SortStrategy s : strategies) {
            if (s == null) continue;
            String name = Optional.ofNullable(s.displayName()).orElse(s.getClass().getSimpleName());
            sortMap.put(name, s);
        }
        for (String name : sortMap.keySet()) {
            model.addElement(name);
        }
        sortCombo.setModel(model);
        sortCombo.setSelectedIndex(0);

        setSortMapper(key -> sortMap.get(key));
    }

    /**
     * Sets a mapper from combo text to {@link SortStrategy} and wires the buttons.
     *
     * @param sortMapper mapping function
     * @throws NullPointerException if sortMapper is null
     */
    public void setSortMapper(Function<String, SortStrategy> sortMapper) {
        this.sortMapper = Objects.requireNonNull(sortMapper, "sortMapper");

        for (var l : sortApplyBtn.getActionListeners()) sortApplyBtn.removeActionListener(l);
        sortApplyBtn.addActionListener(e -> {
            String key = Optional.ofNullable((String) sortCombo.getSelectedItem()).orElse("");
            this.api.setSortStrategy(this.sortMapper.apply(key));
        });

        for (var l : sortResetBtn.getActionListeners()) sortResetBtn.removeActionListener(l);
        sortResetBtn.addActionListener(e -> {
            sortCombo.setSelectedIndex(0);
            String firstKey = (String) sortCombo.getItemAt(0);
            this.api.setSortStrategy(this.sortMapper.apply(firstKey));
        });
    }
```

ToolBox

```java
    /**
     * Sets the supplier used to produce a composed {@link ITaskFilter} for Apply.
     *
     * @param filterSupplier filter supplier
     * @throws NullPointerException if filterSupplier is null
     */
    public void setFilterSupplier(Supplier<ITaskFilter> filterSupplier) {
        this.filterSupplier = Objects.requireNonNull(filterSupplier, "filterSupplier");
        for (var l : filterApplyBtn.getActionListeners()) filterApplyBtn.removeActionListener(l);
        filterApplyBtn.addActionListener(e -> {
            this.api.setFilter(this.filterSupplier.get());
            updateTotalsFromApi();
        });
    }

    /**
     * Binds the Filter UI directly to the API (Apply/Reset/Toggle).
     *
     * @param api the {@link TasksViewAPI} to bind
     * @throws NullPointerException if api is null
     */
    public void bindFilterControls(TasksViewAPI api) {
        this.api = Objects.requireNonNull(api, "api");

        for (var l : filterApplyBtn.getActionListeners()) filterApplyBtn.removeActionListener(l);
        for (var l : filterResetBtn.getActionListeners()) filterResetBtn.removeActionListener(l);
        for (var l : showFilteredTgl.getActionListeners()) showFilteredTgl.removeActionListener(l);

        filterApplyBtn.addActionListener(e -> {
            this.api.setFilter(buildFilterFromUI());
            updateTotalsFromApi();
        });

        filterResetBtn.addActionListener(e -> {
            clearFilterUI();
            this.api.clearFilter();
            updateTotalsFromApi();
        });

        showFilteredTgl.addActionListener(e -> updateTotalsFromApi());

        updateTotalsFromApi();
    }

    private ITaskFilter buildFilterFromUI() {
        ITaskFilter f = Filters.all();

        String q = titleField.getText();
        if (q != null && !q.isBlank()) {
            f = f.and(Filters.titleContains(q.trim()));
        }

        final EnumSet<TaskState> states = EnumSet.noneOf(TaskState.class);
        if (cbTodo.isSelected())       states.add(TaskState.ToDo);
        if (cbInProgress.isSelected()) states.add(TaskState.InProgress);
        if (cbCompleted.isSelected())  states.add(TaskState.Completed);

        if (!states.isEmpty()) {
            ITaskFilter statesFilter = t -> t != null && t.getState() != null && states.contains(t.getState());
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ToolBox.java

ToolBox

```java
            f = f.and(statesFilter);
        }

        return f;
    }

    private void clearFilterUI() {
        titleField.setText("");
        cbTodo.setSelected(false);
        cbInProgress.setSelected(false);
        cbCompleted.setSelected(false);
    }

    /**
     * Sets a custom export handler. If not set, a dialog-based export is used.
     *
     * @param exportHandler handler to execute export
     * @throws NullPointerException if exportHandler is null
     */
    public void setExportHandler(ExportHandler exportHandler) {
        this.exportHandler = Objects.requireNonNull(exportHandler, "exportHandler");
        if (exportBtn != null) {
            for (var l : exportBtn.getActionListeners()) exportBtn.removeActionListener(l);
            exportBtn.addActionListener(e ->
                    this.exportHandler.performExport(this.api, showFilteredTgl.isSelected(), toIdList(safeIds())));
        }
    }

    /**
     * Binds selection property for counters and enables/disables actions accordingly.
     *
     * @param selectionProperty observable selection property (IDs)
     * @throws NullPointerException if selectionProperty is null
     */
    public void bindSelectionProperty(Property<int[]> selectionProperty) {
        this.selectionProp = Objects.requireNonNull(selectionProperty, "selectionProperty");

        if (selectionListener != null) {
            this.selectionProp.removeListener(selectionListener);
        }
        selectionListener = (oldV, newV) -> {
            updateSelectionCount();
            enableActionButtons();
        };
        this.selectionProp.addListener(selectionListener);

        updateSelectionCount();
        enableActionButtons();
    }

    /**
     * Subscribes to tasks and filtered-tasks to keep the Total counter in sync.
     * Safe to call multiple times; replaces previous listeners.
     */
    public void bindTotalsFromApi() {
        if (api == null) return;

        if (tasksListener != null)    api.tasksProperty().removeListener(tasksListener);
        if (filteredListener != null) api.filteredTasksProperty().removeListener(filteredListener);
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ToolBox.java

ToolBox

```java
        tasksListener    = (oldList, newList) -> updateTotalsFromApi();
        filteredListener = (oldList, newList) -> updateTotalsFromApi();

        api.tasksProperty().addListener(tasksListener);
        api.filteredTasksProperty().addListener(filteredListener);

        showFilteredTgl.addActionListener(e -> updateTotalsFromApi());
        updateTotalsFromApi();
    }

    /**
     * Replaces placeholders for Advance/Mark-as with real dialogs and VM calls.
     */
    public void bindAdvanceAndMarkDialogs() {
        for (var l : advanceBtn.getActionListeners()) advanceBtn.removeActionListener(l);
        for (var l : markAsBtn.getActionListeners()) markAsBtn.removeActionListener(l);

        advanceBtn.addActionListener(e -> onAdvance());
        markAsBtn.addActionListener(e -> onMarkAsDialog());
        enableActionButtons();
    }

    private void openExportDialogAndRun() {
        if (api == null) {
            JOptionPane.showMessageDialog(this,
                    "Export is not available: API is not wired.",
                    "Export", JOptionPane.WARNING_MESSAGE);
            return;
        }

        ExportDialog.showDialog(this).ifPresent(res -> {
            Path path = res.file().toPath();
            ExportFormat fmt = res.format();
            boolean onlyFiltered = showFilteredTgl.isSelected();

            try {
                List<Integer> ids = toIdList(safeIds());
                // □ change: don't blindly claim success — verify result by checking file existence
                api.exportTasks(path, fmt, onlyFiltered, ids);
                boolean exists = false;
                try {
                    exists = Files.exists(path);
                } catch (Exception ignore) {
                    // If Files.exists fails for some reason, we'll fall back to generic failure message
                }
                if (!exists) {
                    JOptionPane.showMessageDialog(this,
                            "Export failed: file was not created.\n" + path,
                            "Export", JOptionPane.ERROR_MESSAGE);
                    return;
                }

                JOptionPane.showMessageDialog(this,
                        "Export completed:\n" + path,
                        "Export", JOptionPane.INFORMATION_MESSAGE);
            } catch (Exception ex) {
                JOptionPane.showMessageDialog(this,
                        "Export failed:\n" + ex.getMessage(),
                        "Export", JOptionPane.ERROR_MESSAGE);
            }
        });
    }

    private void onAdvance() {
        int[] ids = safeIds();
        if (ids.length == 0) {
            JOptionPane.showMessageDialog(this, "No tasks selected.", "Advance", JOptionPane.WARNING_MESSAGE);
            return;
        }
        int rc = JOptionPane.showConfirmDialog(
                this,
                "Advance " + ids.length + " selected task(s) to the next state?",
                "Confirm Advance",
                JOptionPane.OK_CANCEL_OPTION,
                JOptionPane.QUESTION_MESSAGE
        );
        if (rc != JOptionPane.OK_OPTION) return;

        // ▯ change: per-task error handling so failures aren't silent
        int failures = 0;
        for (int id : ids) {
            try {
                api.advanceState(id);
            } catch (RuntimeException ex) {
                failures++;
            }
        }
        if (failures > 0) {
            JOptionPane.showMessageDialog(this,
                    "Some tasks failed to advance (" + failures + "). Check logs/DB.",
                    "Advance", JOptionPane.WARNING_MESSAGE);
        }
    }

    private void onMarkAsDialog() {
        int[] ids = safeIds();
        if (ids.length == 0) {
            JOptionPane.showMessageDialog(this, "No tasks selected.", "Mark as…", JOptionPane.WARNING_MESSAGE);
            return;
        }

        final JDialog dlg = new JDialog(SwingUtilities.getWindowAncestor(this), "Mark as…", Dialog.ModalityType.APPLICATION_MODAL);
        dlg.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
        JPanel content = new JPanel(new GridBagLayout());
        content.setBorder(BorderFactory.createEmptyBorder(12, 12, 12, 12));
        content.setBackground(getBackground());

        ButtonGroup bg = new ButtonGroup();
        JRadioButton rbTodo = new JRadioButton("To-Do");
        JRadioButton rbInPr = new JRadioButton("In-Progress");
        JRadioButton rbDone = new JRadioButton("Completed");
        rbTodo.setOpaque(false); rbInPr.setOpaque(false); rbDone.setOpaque(false);
        rbTodo.setSelected(true);
        bg.add(rbTodo); bg.add(rbInPr); bg.add(rbDone);

        JButton ok = new JButton("OK");
        JButton cancel = new JButton("Cancel");
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\taskmanagement\ui\widgets\ToolBox.java

ToolBox

```java
        GridBagConstraints g = new GridBagConstraints();
        g.gridx = 0; g.gridy = 0; g.anchor = GridBagConstraints.WEST; g.insets = new Insets(4,4,4,4);
        content.add(new JLabel("Set state for " + ids.length + " selected task(s):"), g);
        g.gridy++; content.add(rbTodo, g);
        g.gridy++; content.add(rbInPr, g);
        g.gridy++; content.add(rbDone, g);

        JPanel buttons = new JPanel(new FlowLayout(FlowLayout.RIGHT, 8, 0));
        buttons.setOpaque(false);
        buttons.add(ok); buttons.add(cancel);
        g.gridy++; g.anchor = GridBagConstraints.EAST; g.fill = GridBagConstraints.HORIZONTAL; g.weightx = 1.0;
        content.add(buttons, g);

        ok.addActionListener(ev -> {
            TaskState target = rbTodo.isSelected() ? TaskState.ToDo :
                    rbInPr.isSelected() ? TaskState.InProgress : TaskState.Completed;

            boolean warnedBackward = false;

            for (int id : ids) {
                TaskState current = findCurrentState(id);
                if (current == null) continue;

                int curIdx = stateIndex(current);
                int tgtIdx = stateIndex(target);

                if (tgtIdx == curIdx) {
                    continue;
                } else if (tgtIdx > curIdx) {
                    for (int step = curIdx; step < tgtIdx; step++) {
                        try {
                            api.advanceState(id);
                        } catch (RuntimeException ex) {
                            JOptionPane.showMessageDialog(this,
                                    "Failed to advance task #" + id + ": " + ex.getMessage(),
                                    "Mark as…", JOptionPane.ERROR_MESSAGE);
                            break;
                        }
                    }
                } else {
                    if (!warnedBackward) {
                        JOptionPane.showMessageDialog(this,
                                "Backward transitions are not supported by the current workflow.\n" +
                                        "Requested: " + current + " → " + target,
                                "Mark as…", JOptionPane.WARNING_MESSAGE);
                        warnedBackward = true;
                    }
                }
            }

            dlg.dispose();
        });
        cancel.addActionListener(ev -> dlg.dispose());

        dlg.setContentPane(content);
        dlg.pack();
        dlg.setLocationRelativeTo(this);
        dlg.setVisible(true);
```

```java
    }

    private void updateSelectionCount() {
        int sel = 0;
        if (selectionProp != null && selectionProp.getValue() != null) {
            sel = selectionProp.getValue().length;
        } else if (idsProvider != null) {
            sel = safeIds().length;
        }
        selectedCountLbl.setText("Selected: " + sel);
    }

    private void updateTotalsFromApi() {
        if (api == null) return;
        List<?> list = showFilteredTgl.isSelected()
                ? api.filteredTasksProperty().getValue()
                : api.tasksProperty().getValue();
        int total = (list == null) ? 0 : list.size();
        totalCountLbl.setText("Total: " + total);
    }

    private void enableActionButtons() {
        boolean hasSel = false;
        if (selectionProp != null && selectionProp.getValue() != null) {
            hasSel = selectionProp.getValue().length > 0;
        } else if (idsProvider != null) {
            hasSel = safeIds().length > 0;
        }
        advanceBtn.setEnabled(hasSel);
        markAsBtn.setEnabled(hasSel);
    }

    private int[] safeIds() {
        return (idsProvider != null) ? idsProvider.selectedIds() : new int[0];
    }

    private static List<Integer> toIdList(int[] ids) {
        return (ids == null || ids.length == 0)
                ? java.util.List.of()
                : IntStream.of(ids).boxed().toList();
    }

    private static JPanel makeTransparent() {
        JPanel p = new JPanel();
        p.setOpaque(false);
        return p;
    }

    private static JPanel centerInGridBag(JComponent c) {
        JPanel wrap = makeTransparent();
        wrap.setLayout(new GridBagLayout());
        wrap.add(c, new GridBagConstraints());
        return wrap;
    }

    private void styleMiniRound(JButton b, Color bg, Color fg) {
        b.setFocusPainted(false);
        b.setOpaque(true);
        b.setBackground(bg);
        b.setForeground(fg);
        b.setFont(AppTheme.TB_LABEL_FONT_LG);
        b.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(AppTheme.TB_FIELD_BORDER, 1, true),
                BorderFactory.createEmptyBorder(8, 14, 8, 14)
        ));
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    }

    private void styleIconTextButton(JButton b, Icon icon, Color bg, Color fg) {
        b.setIcon(icon);
        b.setFocusPainted(false);
        b.setOpaque(true);
        b.setBackground(bg);
        b.setForeground(fg);
        b.setFont(AppTheme.TB_LABEL_FONT_LG);
        b.setHorizontalTextPosition(SwingConstants.CENTER);
        b.setVerticalTextPosition(SwingConstants.BOTTOM);
        b.setBorder(BorderFactory.createCompoundBorder(
                BorderFactory.createLineBorder(AppTheme.TB_FIELD_BORDER, 1, true),
                BorderFactory.createEmptyBorder(8, 16, 8, 16)
        ));
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    }

    private void styleSmallFilled(AbstractButton b, Color bg, Color fg) {
        b.setFocusPainted(false);
        b.setOpaque(true);
        b.setBackground(bg);
        b.setForeground(fg);
        b.setFont(AppTheme.TB_LABEL_FONT_LG);
        b.setBorder(BorderFactory.createCompoundBorder(
                new javax.swing.border.LineBorder(bg.darker(), 1, true),
                BorderFactory.createEmptyBorder(4, 6, 4, 6)
        ));
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
    }

    private void styleSmallHollow(AbstractButton b,
                                  Color border, Color fg,
                                  Color selectedBg, Color selectedFg) {
        b.setFocusPainted(false);
        b.setOpaque(false);
        b.setForeground(fg);
        b.setFont(AppTheme.TB_LABEL_FONT_LG);
        b.setBorder(BorderFactory.createCompoundBorder(
                new javax.swing.border.LineBorder(border, 1, true),
                BorderFactory.createEmptyBorder(4, 6, 4, 6)
        ));
        b.setCursor(Cursor.getPredefinedCursor(Cursor.HAND_CURSOR));
        b.setBackground(new Color(0, 0, 0, 0));
        if (b instanceof JToggleButton tgl) {
            tgl.addItemListener(e -> {
                boolean on = tgl.isSelected();
                tgl.setOpaque(on);
                tgl.setBackground(on ? selectedBg : new Color(0, 0, 0, 0));
                tgl.setForeground(on ? selectedFg : fg);
            });
        }
    }

    private static Icon safeIcon(Icon icon) {
        if (icon != null) return icon;
        return new Icon() {
            @Override public void paintIcon(Component c, Graphics g, int x, int y) { }
            @Override public int getIconWidth() { return 1; }
            @Override public int getIconHeight() { return 1; }
        };
    }

    private void styleCheck(JCheckBox cb) {
        cb.setOpaque(false);
        cb.setForeground(AppTheme.TB_TEXT_FG);
        cb.setFocusPainted(false);
        cb.setBorder(BorderFactory.createEmptyBorder(2, 2, 2, 2));
        cb.setFont(AppTheme.TB_RADIO_FONT);
    }

    /**
     * Updates counters text. Typically invoked automatically via bindings.
     *
     * @param selected number of selected tasks (non-negative)
     * @param total    total number of tasks (non-negative)
     */
    public void updateCounters(int selected, int total) {
        selectedCountLbl.setText("Selected: " + Math.max(0, selected));
        totalCountLbl.setText("Total: " + Math.max(0, total));
    }

    /** @return the Undo button component. */
    public JButton getUndoButton() { return undoBtn; }
    /** @return the Redo button component. */
    public JButton getRedoButton() { return redoBtn; }
    /** @return the Advance button component. */
    public JButton getAdvanceButton() { return advanceBtn; }
    /** @return the Mark-as button component. */
    public JButton getMarkAsButton() { return markAsBtn; }
    /** @return the sort combo box. */
    public JComboBox<String> getSortCombo() { return sortCombo; }
    /** @return the Sort Apply button. */
    public JButton getSortApplyButton() { return sortApplyBtn; }
    /** @return the Sort Reset button. */
    public JButton getSortResetButton() { return sortResetBtn; }
    /** @return the To-Do checkbox. */
    public JCheckBox getCbTodo() { return cbTodo; }
    /** @return the In-Progress checkbox. */
    public JCheckBox getCbInProgress() { return cbInProgress; }
    /** @return the Completed checkbox. */
    public JCheckBox getCbCompleted() { return cbCompleted; }
    /** @return the Filter Apply button. */
    public JButton getFilterApplyButton() { return filterApplyBtn; }
    /** @return the Filter Reset button. */
    public JButton getFilterResetButton() { return filterResetBtn; }
    /** @return the toggle that counts filtered as total. */
    public JToggleButton getShowFilteredToggle() { return showFilteredTgl; }
    /** @return the Selected counter label. */
    public JLabel getSelectedCountLabel() { return selectedCountLbl; }
    /** @return the Total counter label. */
    public JLabel getTotalCountLabel() { return totalCountLbl; }
    /** @return the Export button. */
    public JButton getExportButton() { return exportBtn; }

    /**
     * Supplies currently selected task IDs.
     */
    @FunctionalInterface
    public interface IdsProvider {
        /**
         * @return selected task IDs (empty if none)
         */
        int[] selectedIds();
    }

    /**
     * Handles exporting tasks according to user choices.
     */
    @FunctionalInterface
    public interface ExportHandler {
        /**
         * Performs the export operation.
         *
         * @param api          bound {@link TasksViewAPI}
         * @param useFiltered  whether to export filtered list
         * @param selectedIds  selected task IDs to include (may be empty)
         */
        void performExport(TasksViewAPI api, boolean useFiltered, List<Integer> selectedIds);
    }

    private TaskState findCurrentState(int id) {
        if (api == null) return null;
        java.util.List<ITask> list = getPreferredList();
        if (list != null) {
            for (ITask t : list) if (t.getId() == id) return t.getState();
        }
        list = showFilteredTgl.isSelected()
                ? api.filteredTasksProperty().getValue()
                : api.tasksProperty().getValue();
        if (list != null) {
            for (ITask t : list) if (t.getId() == id) return t.getState();
        }
        return null;
    }

    private java.util.List<ITask> getPreferredList() {
        return showFilteredTgl.isSelected()
                ? api.filteredTasksProperty().getValue()
                : api.tasksProperty().getValue();
    }

    private static int stateIndex(TaskState s) {
        return switch (s) {
            case ToDo -> 0;
            case InProgress -> 1;
            case Completed -> 2;
        };
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\application\viewmodel\commands\CommandStackTest.java

CommandStackTest

```java
package taskmanagement.application.viewmodel.commands;

import org.junit.Before;
import org.junit.Test;
import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.persistence.ITasksDAO;
import taskmanagement.persistence.TasksDAOException;

import java.util.*;
import java.util.concurrent.atomic.AtomicInteger;

import static org.junit.Assert.*;

/**
 * JUnit 4 tests for the command stack (execute/undo/redo) used in the
 * tasks management application. Verifies behavior of {@code AddTaskCommand},
 * {@code UpdateTaskCommand}, {@code DeleteTaskCommand}, and {@code MarkStateCommand}
 * against an in-memory fake DAO using a minimal mutable task double.
 */
public final class CommandStackTest {

    private static final class MutableTask implements ITask {
        private int id;
        private String title;
        private String description;
        private TaskState state;

        MutableTask(int id, String title, String description, TaskState state) {
            this.id = id;
            this.title = Objects.requireNonNull(title, "title");
            this.description = Objects.requireNonNull(description, "description");
            this.state = Objects.requireNonNull(state, "state");
        }

        MutableTask(MutableTask other) {
            this(other.id, other.title, other.description, other.state);
        }

        void setId(int id) { this.id = id; }
        void setTitle(String t) { this.title = Objects.requireNonNull(t); }
        void setDescription(String d) { this.description = Objects.requireNonNull(d); }
        void setState(TaskState s) { this.state = Objects.requireNonNull(s); }

        @Override public int getId() { return id; }
        @Override public String getTitle() { return title; }
        @Override public String getDescription() { return description; }
        @Override public TaskState getState() { return state; }

        @Override
        public void accept(taskmanagement.domain.visitor.TaskVisitor visitor) {
            // Intentionally unused in these tests.
        }
    }

    private static final class FakeDAO implements ITasksDAO {
        private final Map<Integer, ITask> store = new LinkedHashMap<>();
        private final AtomicInteger seq = new AtomicInteger(1);
```

CommandStackTest

```java
        @Override
        public ITask[] getTasks() { return store.values().toArray(new ITask[0]); }

        @Override
        public ITask getTask(int id) throws TasksDAOException {
            ITask t = store.get(id);
            if (t == null) throw new TasksDAOException("Not found: " + id);
            return t;
        }

        @Override
        public void addTask(ITask task) throws TasksDAOException {
            if (!(task instanceof MutableTask mt)) {
                throw new TasksDAOException("Test expects MutableTask for id reflection");
            }
            int id = mt.getId();
            if (id <= 0) {
                id = seq.getAndIncrement();
                mt.setId(id);
            } else {
                if (store.containsKey(id)) {
                    throw new TasksDAOException("Duplicate id: " + id);
                }
            }
            store.put(id, cloneForSafety(mt));
        }

        @Override
        public void updateTask(ITask task) throws TasksDAOException {
            if (!(task instanceof MutableTask mt)) {
                throw new TasksDAOException("Test expects MutableTask");
            }
            int id = mt.getId();
            if (!store.containsKey(id)) throw new TasksDAOException("Cannot update missing id: " + id);
            store.put(id, cloneForSafety(mt));
        }

        @Override
        public void deleteTasks() { store.clear(); }

        @Override
        public void deleteTask(int id) throws TasksDAOException {
            if (store.remove(id) == null) throw new TasksDAOException("Nothing to delete for id: " + id);
        }

        private static MutableTask cloneForSafety(MutableTask mt) {
            return new MutableTask(mt.getId(), mt.getTitle(), mt.getDescription(), mt.getState());
        }

        boolean exists(int id) { return store.containsKey(id); }
        Optional<MutableTask> find(int id) {
            ITask t = store.get(id);
            return Optional.ofNullable(t).map(it -> new MutableTask((MutableTask) it));
        }
        int size() { return store.size(); }
    }

    private FakeDAO dao;
    private CommandStack stack;
```

CommandStackTest

```java
    /**
     * Initializes the in-memory DAO and command stack before each test.
     */
    @Before
    public void setUp() {
        dao = new FakeDAO();
        stack = new CommandStack();
    }

    /**
     * Ensures {@link AddTaskCommand} adds a row and reflects an assigned id,
     * then validates that undo deletes the row and redo reinserts it with the same id.
     *
     * @throws Exception if the command execution fails unexpectedly
     */
    @Test
    public void add_execute_undo_redo() throws Exception {
        MutableTask t = new MutableTask(0, "A", "desc", TaskState.ToDo);
        AddTaskCommand add = new AddTaskCommand(dao, t);

        stack.execute(add);
        assertTrue("id should be assigned (>0)", t.getId() > 0);
        int assigned = t.getId();
        assertEquals(1, dao.size());
        assertTrue(dao.exists(assigned));

        stack.undo();
        assertEquals(0, dao.size());
        assertFalse(dao.exists(assigned));

        stack.redo();
        assertEquals(1, dao.size());
        assertTrue("redo should reinsert same id", dao.exists(assigned));
    }

    /**
     * Ensures {@link UpdateTaskCommand} applies the updated snapshot on execute,
     * restores the original snapshot on undo, and reapplies the update on redo.
     *
     * @throws Exception if the command execution fails unexpectedly
     */
    @Test
    public void update_execute_undo_redo() throws Exception {
        MutableTask before = new MutableTask(0, "Title-1", "D", TaskState.ToDo);
        new AddTaskCommand(dao, before).execute();
        int id = before.getId();

        MutableTask after = new MutableTask(id, "Title-2", "D", TaskState.ToDo);
        UpdateTaskCommand upd = new UpdateTaskCommand(dao, copy(before), copy(after));

        stack.execute(upd);
        assertEquals("Title-2", dao.getTask(id).getTitle());

        stack.undo();
        assertEquals("Title-1", dao.getTask(id).getTitle());

        stack.redo();
        assertEquals("Title-2", dao.getTask(id).getTitle());
    }

    /**
     * Ensures {@link DeleteTaskCommand} removes a row on execute, restores it
     * with the same id on undo, and removes it again on redo.
     *
     * @throws Exception if the command execution fails unexpectedly
     */
    @Test
    public void delete_execute_undo_redo() throws Exception {
        MutableTask toDelete = new MutableTask(0, "X", "D", TaskState.ToDo);
        new AddTaskCommand(dao, toDelete).execute();
        int id = toDelete.getId();
        assertTrue(dao.exists(id));

        DeleteTaskCommand del = new DeleteTaskCommand(dao, copy(toDelete));

        stack.execute(del);
        assertFalse(dao.exists(id));

        stack.undo();
        assertTrue("undo should bring task back", dao.exists(id));

        stack.redo();
        assertFalse(dao.exists(id));
    }

    /**
     * Ensures {@link MarkStateCommand} changes state on execute, restores the
     * previous state on undo, and reapplies the new state on redo.
     *
     * @throws Exception if the command execution fails unexpectedly
     */
    @Test
    public void mark_state_execute_undo_redo() throws Exception {
        MutableTask t = new MutableTask(0, "S", "D", TaskState.ToDo);
        new AddTaskCommand(dao, t).execute();
        int id = t.getId();
        assertEquals(TaskState.ToDo, dao.getTask(id).getState());

        MarkStateCommand.TaskFactory factory = (src, newState) -> {
            MutableTask srcMt = (MutableTask) src;
            MutableTask copy = new MutableTask(srcMt);
            copy.setState(newState);
            return copy;
        };

        MarkStateCommand mark = new MarkStateCommand(dao, copy(t), TaskState.InProgress, factory);

        stack.execute(mark);
        assertEquals(TaskState.InProgress, dao.getTask(id).getState());

        stack.undo();
        assertEquals(TaskState.ToDo, dao.getTask(id).getState());

        stack.redo();
        assertEquals(TaskState.InProgress, dao.getTask(id).getState());
    }
```

CommandStackTest

```java
    private static MutableTask copy(MutableTask mt) {
        return new MutableTask(mt.getId(), mt.getTitle(), mt.getDescription(), mt.getState());
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\application\viewmodel\events\ObserverPropertyTest.java

ObserverPropertyTest

```java
package taskmanagement.application.viewmodel.events;

import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.List;
import java.util.Objects;
import java.util.concurrent.atomic.AtomicInteger;

import static org.junit.Assert.*;

/**
 * JUnit 4 test suite for the ViewModel observer utilities
 * {@code Property<T>} and {@code ObservableList<T>}.
 * <p>
 * Verifies listener notification semantics, conditional updates,
 * multi-listener behavior, listener removal, and robustness in the
 * presence of listener exceptions.
 */
public final class ObserverPropertyTest {

    private Property<String> prop;
    private ObservableList<Integer> olist;

    /**
     * Initializes the property and observable list before each test.
     */
    @Before
    public void setUp() {
        prop = new Property<>("A");
        olist = new ObservableList<>();
    }

    /**
     * Ensures {@code Property#setValue} always notifies listeners,
     * including when the new value equals the current value.
     */
    @Test
    public void property_setValue_alwaysNotifies_evenIfEqual() {
        final AtomicInteger calls = new AtomicInteger(0);
        final List<String> last = new ArrayList<>(2);

        Property.Listener<String> l = (oldV, newV) -> {
            calls.incrementAndGet();
            last.clear();
            last.add(oldV);
            last.add(newV);
        };

        prop.addListener(l);

        prop.setValue("B");
        assertEquals(1, calls.get());
        assertEquals(List.of("A", "B"), last);

        prop.setValue("B");
        assertEquals(2, calls.get());
        assertEquals(List.of("B", "B"), last);
    }

    /**
     * Ensures {@code Property#setValueIfChanged} notifies listeners
     * only when the value actually changes.
     */
    @Test
    public void property_setValueIfChanged_notifiesOnlyWhenDifferent() {
        final AtomicInteger calls = new AtomicInteger(0);
        prop.addListener((oldV, newV) -> calls.incrementAndGet());

        prop.setValueIfChanged("A");
        assertEquals(0, calls.get());

        prop.setValueIfChanged("Z");
        assertEquals(1, calls.get());

        prop.setValueIfChanged("Z");
        assertEquals(1, calls.get());
    }

    /**
     * Ensures {@code Property#fireChange()} notifies listeners using the
     * current value as both old and new.
     */
    @Test
    public void property_fireChange_notifiesWithSameValue() {
        final List<String> pairs = new ArrayList<>();
        prop.addListener((oldV, newV) -> pairs.add(oldV + "→" + newV));

        prop.fireChange();
        assertEquals(1, pairs.size());
        assertEquals("A→A", pairs.get(0));
    }

    /**
     * Verifies that removing a listener prevents further notifications.
     */
    @Test
    public void property_removeListener_noFurtherNotifications() {
        final AtomicInteger calls = new AtomicInteger();
        Property.Listener<String> l = (o, n) -> calls.incrementAndGet();
        prop.addListener(l);

        prop.setValue("X");
        assertEquals(1, calls.get());

        prop.removeListener(l);
        prop.setValue("Y");
        assertEquals(1, calls.get());
    }

    /**
     * Verifies multiple listeners are invoked even if one throws,
     * and that exceptions from one listener do not prevent others.
     */
    @Test
    public void property_multipleListeners_allAreCalled_evenIfOneThrows() {
        final AtomicInteger calls = new AtomicInteger();
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\application\viewmodel\events\ObserverPropertyTest.java

ObserverPropertyTest

```java
        prop.addListener((o, n) -> calls.incrementAndGet());
        prop.addListener((o, n) -> { throw new RuntimeException("boom"); });
        prop.addListener((o, n) -> calls.incrementAndGet());

        prop.setValue("B");
        assertEquals(2, calls.get());
    }

    /**
     * Ensures {@code ObservableList#set} notifies listeners only when the
     * snapshot content changes.
     */
    @Test
    public void olist_set_notifiesOnlyOnRealChange() {
        final AtomicInteger calls = new AtomicInteger(0);
        final List<List<Integer>> snapshots = new ArrayList<>();
        olist.addListener(newSnap -> {
            calls.incrementAndGet();
            snapshots.add(newSnap);
        });

        olist.set(List.of());
        assertEquals(0, calls.get());

        olist.set(List.of(1, 2, 3));
        assertEquals(1, calls.get());
        assertEquals(List.of(1, 2, 3), snapshots.get(0));

        olist.set(List.of(1, 2, 3));
        assertEquals(1, calls.get());

        olist.set(List.of(1, 2, 3, 4));
        assertEquals(2, calls.get());
        assertEquals(List.of(1, 2, 3, 4), snapshots.get(1));
    }

    /**
     * Ensures {@code ObservableList#clear} notifies only if the list
     * was previously non-empty.
     */
    @Test
    public void olist_clear_notifiesOnlyIfWasNonEmpty() {
        final AtomicInteger calls = new AtomicInteger();
        olist.addListener(newSnap -> calls.incrementAndGet());

        olist.clear();
        assertEquals(0, calls.get());

        olist.set(List.of(9));
        assertEquals(1, calls.get());
        olist.clear();
        assertEquals(2, calls.get());
    }

    /**
     * Verifies adding and removing listeners affects notification delivery.
     */
    @Test
```

ObserverPropertyTest

```java
    public void olist_addRemoveListeners() {
        final AtomicInteger calls = new AtomicInteger();
        ObservableList.Listener<Integer> l = newSnap -> calls.incrementAndGet();

        olist.addListener(l);
        olist.set(List.of(1));
        assertEquals(1, calls.get());

        olist.removeListener(l);
        olist.set(List.of(2));
        assertEquals(1, calls.get());
    }

    /**
     * Verifies multiple list listeners are invoked even if one throws.
     */
    @Test
    public void olist_multipleListeners_allAreCalled_evenIfOneThrows() {
        final AtomicInteger calls = new AtomicInteger();
        olist.addListener(newSnap -> calls.incrementAndGet());
        olist.addListener(newSnap -> { throw new RuntimeException("boom"); });
        olist.addListener(newSnap -> calls.incrementAndGet());

        olist.set(List.of(1, 2));
        assertEquals(2, calls.get());
    }

    /**
     * Small helper that compares lists by content using {@link Objects#equals(Object, Object)}.
     *
     * @param a first list
     * @param b second list
     * @param <T> element type
     * @return {@code true} if equal by content, otherwise {@code false}
     */
    private static <T> boolean equalLists(List<T> a, List<T> b) {
        return Objects.equals(a, b);
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\application\viewmodel\sort\StrategyTest.java

StrategyTest

```java
package taskmanagement.application.viewmodel.sort;

import org.junit.Test;
import taskmanagement.domain.ITask;
import taskmanagement.domain.TaskState;
import taskmanagement.domain.visitor.TaskVisitor;

import java.lang.reflect.Method;
import java.util.*;
import java.util.stream.Collectors;

import static org.junit.Assert.*;

/**
 * JUnit 4 tests for sorting strategies: {@code SortById} (ascending by id),
 * {@code SortByTitle} (case-insensitive lexicographic order), and
 * {@code SortByState} (ToDo &lt; InProgress &lt; Completed).
 * <p>
 * The suite is compatible with multiple strategy API shapes:
 * a strategy may implement {@link Comparator}, expose {@code sort(List&lt;ITask&gt;)},
 * or expose {@code comparator()}.
 */
public final class StrategyTest {

    private static final class T implements ITask {
        private final int id;
        private final String title;
        private final String description;
        private final TaskState state;

        T(int id, String title, String description, TaskState state) {
            this.id = id;
            this.title = Objects.requireNonNull(title);
            this.description = Objects.requireNonNull(description);
            this.state = Objects.requireNonNull(state);
        }

        @Override public int getId() { return id; }
        @Override public String getTitle() { return title; }
        @Override public String getDescription() { return description; }
        @Override public TaskState getState() { return state; }
        @Override public void accept(TaskVisitor visitor) { }

        @Override public String toString() { return "T{id=" + id + ", title='" + title + "', state=" + state + '}'; }
    }

    @SuppressWarnings({ "unchecked", "rawtypes" })
    private static List<ITask> apply(SortStrategy strategy, List<ITask> src) {
        if (strategy instanceof Comparator) {
            List<ITask> copy = new ArrayList<>(src);
            copy.sort((Comparator) strategy);
            return copy;
        }
        try {
            Method m = strategy.getClass().getMethod("sort", List.class);
            Object out = m.invoke(strategy, new ArrayList<>(src));
            if (out instanceof List) return (List<ITask>) out;
        } catch (NoSuchMethodException ignore) {
        } catch (ReflectiveOperationException e) {
```

StrategyTest

```java
            throw new AssertionError("Failed invoking sort(List): " + e.getMessage(), e);
        }
        try {
            Method m = strategy.getClass().getMethod("comparator");
            Object cmp = m.invoke(strategy);
            if (cmp instanceof Comparator) {
                List<ITask> copy = new ArrayList<>(src);
                copy.sort((Comparator<? super ITask>) cmp);
                return copy;
            }
        } catch (NoSuchMethodException ignore) {
        } catch (ReflectiveOperationException e) {
            throw new AssertionError("Failed invoking comparator(): " + e.getMessage(), e);
        }
        throw new AssertionError("Unknown SortStrategy API: " + strategy.getClass().getName());
    }

    private static List<Integer> ids(List<ITask> list)   { return list.stream().map(ITask::getId).collect(Collectors.toList()); }
    private static List<String>  titles(List<ITask> list) { return list.stream().map(ITask::getTitle).collect(Collectors.toList()); }
    private static List<TaskState> states(List<ITask> list){ return list.stream().map(ITask::getState).collect(Collectors.toList()); }

    private static List<ITask> sample() {
        return List.of(
                new T(3, "Bravo",   "b", TaskState.InProgress),
                new T(1, "alpha",   "a", TaskState.ToDo),
                new T(2, "Charlie","c", TaskState.Completed),
                new T(5, "alpha",   "x", TaskState.Completed),
                new T(4, "delta",   "d", TaskState.ToDo)
        );
    }

    /**
     * Verifies that {@link SortById} orders tasks by ascending id.
     */
    @Test
    public void sortById_ascending() {
        SortStrategy s = new SortById();
        List<ITask> sorted = apply(s, sample());
        assertEquals(List.of(1, 2, 3, 4, 5), ids(sorted));
    }

    /**
     * Verifies that {@link SortById} is stable for empty and single-element inputs.
     */
    @Test
    public void sortById_emptyAndSingle_areStable() {
        SortStrategy s = new SortById();
        assertTrue(apply(s, List.of()).isEmpty());
        ITask one = new T(7, "x", "d", TaskState.ToDo);
        assertEquals(List.of(one), apply(s, List.of(one)));
    }

    /**
     * Verifies that {@link SortByTitle} sorts titles case-insensitively in lexicographic order.
     */
    @Test
    public void sortByTitle_lexicographic_caseInsensitive() {
        SortStrategy s = new SortByTitle();
        List<ITask> sorted = apply(s, sample());
```

StrategyTest

```java
        List<String> actual = titles(sorted);

        List<String> expected = new ArrayList<>(titles(sample()));
        expected.sort(String.CASE_INSENSITIVE_ORDER);

        assertEquals("titles should be sorted case-insensitively", expected, actual);

        int i = actual.indexOf("alpha"), j = actual.lastIndexOf("alpha");
        assertTrue("alphas should be adjacent", j - i == 1);
    }

    /**
     * Verifies that {@link SortByTitle} is stable for empty and single-element inputs.
     */
    @Test
    public void sortByTitle_emptyAndSingle_areStable() {
        SortStrategy s = new SortByTitle();
        assertTrue(apply(s, List.of()).isEmpty());
        ITask one = new T(9, "Only", "d", TaskState.Completed);
        assertEquals(List.of("Only"), titles(apply(s, List.of(one))));
    }

    /**
     * Verifies that {@link SortByState} orders tasks by lifecycle:
     * ToDo, then InProgress, then Completed.
     */
    @Test
    public void sortByState_order_ToDo_InProgress_Completed() {
        SortStrategy s = new SortByState();
        List<ITask> sorted = apply(s, sample());
        List<TaskState> st = states(sorted);

        int firstIP = st.indexOf(TaskState.InProgress);
        int firstC  = st.indexOf(TaskState.Completed);

        for (int k = 0; k < firstIP; k++)            assertEquals(TaskState.ToDo, st.get(k));
        for (int k = firstIP; k < firstC; k++)       assertEquals(TaskState.InProgress, st.get(k));
        for (int k = firstC; k < st.size(); k++)     assertEquals(TaskState.Completed, st.get(k));
    }

    /**
     * Verifies that {@link SortByState} is stable for empty and single-element inputs.
     */
    @Test
    public void sortByState_emptyAndSingle_areStable() {
        SortStrategy s = new SortByState();
        assertTrue(apply(s, List.of()).isEmpty());
        ITask one = new T(11, "one", "d", TaskState.InProgress);
        assertEquals(List.of(TaskState.InProgress), states(apply(s, List.of(one))));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\domain\FiltersTest.java

FiltersTest

```java
package taskmanagement.domain;

import org.junit.*;
import taskmanagement.domain.filter.Filters;
import taskmanagement.domain.filter.ITaskFilter;

/**
 * JUnit 4 tests for the composable task {@link Filters} and the
 * {@link ITaskFilter} combinators (AND/OR/NOT/ALL) against the domain model.
 */
public class FiltersTest {

    private Task t1, t2, t3;

    /**
     * Initializes sample tasks used across the test cases.
     */
    @Before
    public void setUp() {
        t1 = new Task(1, "Write tests", "DAO CRUD", TaskState.ToDo);
        t2 = new Task(2, "Wire UI", "MVVM binding", TaskState.InProgress);
        t3 = new Task(3, "Polish UX", "Dark theme", TaskState.Completed);
    }

    /**
     * Verifies {@link Filters#titleContains(String)} performs a case-insensitive
     * containment match on task titles.
     */
    @Test
    public void titleContains_basic() {
        Assert.assertTrue(Filters.titleContains("write").test(t1));
        Assert.assertTrue(Filters.titleContains("WRITE").test(t1));
        Assert.assertFalse(Filters.titleContains("xyz").test(t1));
    }

    /**
     * Verifies {@link Filters#descriptionContains(String)} performs a
     * case-insensitive containment match on task descriptions.
     */
    @Test
    public void descriptionContains_basic() {
        Assert.assertTrue(Filters.descriptionContains("crud").test(t1));
        Assert.assertTrue(Filters.descriptionContains("BIND").test(t2));
        Assert.assertFalse(Filters.descriptionContains("nope").test(t3));
    }

    /**
     * Verifies {@link Filters#idEquals(int)} matches only the specified id.
     */
    @Test
    public void idEquals_basic() {
        Assert.assertTrue(Filters.idEquals(2).test(t2));
        Assert.assertFalse(Filters.idEquals(99).test(t2));
    }

    /**
     * Verifies {@link Filters#stateIs(TaskState)} and the alias
     * {@link Filters#byState(TaskState)} for state-based matching.
     */
```

FiltersTest

```java
    @Test
    public void stateIs_and_alias() {
        Assert.assertTrue(Filters.stateIs(TaskState.Completed).test(t3));
        Assert.assertTrue(Filters.byState(TaskState.ToDo).test(t1));
        Assert.assertFalse(Filters.byState(TaskState.Completed).test(t1));
    }

    /**
     * Verifies logical composition via {@link ITaskFilter#and(ITaskFilter)},
     * {@link ITaskFilter#or(ITaskFilter)}, {@link ITaskFilter#not(ITaskFilter)},
     * and the match-all predicate {@link Filters#all()}.
     */
    @Test
    public void combinator_and_or_not() {
        ITaskFilter byTitle = Filters.titleContains("ui");
        ITaskFilter inProg  = Filters.stateIs(TaskState.InProgress);

        Assert.assertTrue(byTitle.or(inProg).test(t2));
        Assert.assertFalse(byTitle.and(inProg).test(t1));
        Assert.assertTrue(ITaskFilter.not(byTitle).test(t1));
        Assert.assertTrue(Filters.all().test(t1));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\domain\StateTransitionTest.java

StateTransitionTest

```java
package taskmanagement.domain;

import org.junit.Assert;
import org.junit.Test;
import taskmanagement.domain.exceptions.ValidationException;

/**
 * JUnit 4 tests verifying the {@link Task} lifecycle transitions (State pattern).
 * <p>Allowed transitions: ToDo → InProgress → Completed.</p>
 * <p>Forbidden transitions: backward moves (InProgress → ToDo, Completed → InProgress/ToDo).</p>
 * <p>Idempotency: setting the same state twice is allowed.</p>
 * <p>Validation: setting {@code null} state throws {@link ValidationException}.</p>
 */
public class StateTransitionTest {

    private static Task newTaskIn(TaskState state) {
        return new Task(200, "Sample", "Lifecycle test", state);
    }

    /**
     * Allows transition from {@link TaskState#ToDo} to {@link TaskState#InProgress}.
     *
     * @throws Exception if test execution fails unexpectedly
     */
    @Test
    public void allow_todo_to_inprogress() throws Exception {
        Task task = newTaskIn(TaskState.ToDo);
        task.setState(TaskState.InProgress);
        Assert.assertEquals(TaskState.InProgress, task.getState());
    }

    /**
     * Allows transition from {@link TaskState#InProgress} to {@link TaskState#Completed}.
     *
     * @throws Exception if test execution fails unexpectedly
     */
    @Test
    public void allow_inprogress_to_completed() throws Exception {
        Task task = newTaskIn(TaskState.InProgress);
        task.setState(TaskState.Completed);
        Assert.assertEquals(TaskState.Completed, task.getState());
    }

    /**
     * Allows reaching {@link TaskState#Completed} from {@link TaskState#ToDo} via two steps.
     *
     * @throws Exception if test execution fails unexpectedly
     */
    @Test
    public void allow_todo_to_completed_via_two_steps() throws Exception {
        Task task = newTaskIn(TaskState.ToDo);
        task.setState(TaskState.InProgress);
        task.setState(TaskState.Completed);
        Assert.assertEquals(TaskState.Completed, task.getState());
    }

    /**
     * Forbids backward transition from {@link TaskState#InProgress} to {@link TaskState#ToDo}.
     *
```

StateTransitionTest

```
     * @throws Exception always thrown by the tested operation
     */
    @Test(expected = ValidationException.class)
    public void forbid_inprogress_back_to_todo() throws Exception {
        Task task = newTaskIn(TaskState.InProgress);
        task.setState(TaskState.ToDo);
    }

    /**
     * Forbids backward transition from {@link TaskState#Completed} to {@link TaskState#InProgress}.
     *
     * @throws Exception always thrown by the tested operation
     */
    @Test(expected = ValidationException.class)
    public void forbid_completed_back_to_inprogress() throws Exception {
        Task task = newTaskIn(TaskState.Completed);
        task.setState(TaskState.InProgress);
    }

    /**
     * Forbids backward transition from {@link TaskState#Completed} to {@link TaskState#ToDo}.
     *
     * @throws Exception always thrown by the tested operation
     */
    @Test(expected = ValidationException.class)
    public void forbid_completed_back_to_todo() throws Exception {
        Task task = newTaskIn(TaskState.Completed);
        task.setState(TaskState.ToDo);
    }

    /**
     * Permits setting the same state value repeatedly (idempotent behavior).
     *
     * @throws Exception if test execution fails unexpectedly
     */
    @Test
    public void idempotent_set_same_state() throws Exception {
        Task t1 = newTaskIn(TaskState.ToDo);
        t1.setState(TaskState.ToDo);
        Assert.assertEquals(TaskState.ToDo, t1.getState());

        Task t2 = newTaskIn(TaskState.InProgress);
        t2.setState(TaskState.InProgress);
        Assert.assertEquals(TaskState.InProgress, t2.getState());

        Task t3 = newTaskIn(TaskState.Completed);
        t3.setState(TaskState.Completed);
        Assert.assertEquals(TaskState.Completed, t3.getState());
    }

    /**
     * Ensures {@code null} state assignment is invalid and results in {@link ValidationException}.
     *
     * @throws Exception always thrown by the tested operation
     */
    @Test(expected = ValidationException.class)
    public void null_state_is_invalid() throws Exception {
        Task task = newTaskIn(TaskState.ToDo);
        task.setState(null);
```

StateTransitionTest

```
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\domain\visitor\VisitorReportTest.java

VisitorReportTest

```java
package taskmanagement.domain.visitor;

import org.junit.Assert;
import org.junit.Test;

import taskmanagement.domain.Task;
import taskmanagement.domain.TaskState;

import taskmanagement.domain.visitor.export.CsvFlatTaskVisitor;
import taskmanagement.domain.visitor.export.PlainTextFlatTaskVisitor;

import taskmanagement.domain.visitor.reports.Report;
import taskmanagement.domain.visitor.reports.ByStateCount;
import taskmanagement.domain.visitor.adapters.IReportExporter;
import taskmanagement.domain.visitor.adapters.ByStateCsvExporter;
import taskmanagement.domain.visitor.adapters.ByStatePlainTextExporter;

/**
 * JUnit 4 tests for the visitor-based reporting and exporting features.
 * <p>
 * Verifies flat CSV/plain-text export via {@link Task#accept(taskmanagement.domain.visitor.TaskVisitor)}
 * and validates counting by task state with subsequent export through concrete exporters.
 */
public class VisitorReportTest {

    /**
     * Verifies CSV export using the flat visitor routed via {@code Task.accept(visitor)}.
     */
    @Test
    public void csv_export_via_accept() {
        Task t1 = new Task(1, "Write tests", "DAO CRUD", TaskState.ToDo);
        Task t2 = new Task(2, "Wire UI",     "MVVM binding", TaskState.InProgress);
        Task t3 = new Task(3, "Polish UX",   "Dark theme",   TaskState.Completed);

        CsvFlatTaskVisitor csv = new CsvFlatTaskVisitor();
        t1.accept(csv);
        t2.accept(csv);
        t3.accept(csv);
        csv.complete();

        String out = csv.result();
        Assert.assertTrue(out.startsWith("id,title,description,state\n"));
        Assert.assertTrue(out.contains("1,\"Write tests\",\"DAO CRUD\",ToDo"));
        Assert.assertTrue(out.contains("2,\"Wire UI\",\"MVVM binding\",InProgress"));
        Assert.assertTrue(out.contains("3,\"Polish UX\",\"Dark theme\",Completed"));
    }

    /**
     * Verifies plain-text export using the flat visitor routed via {@code Task.accept(visitor)}.
     */
    @Test
    public void plaintext_export_via_accept() {
        Task t1 = new Task(10, "A", "a", TaskState.ToDo);
        Task t2 = new Task(11, "B", "b", TaskState.InProgress);
        Task t3 = new Task(12, "C", "c", TaskState.Completed);

        PlainTextFlatTaskVisitor txt = new PlainTextFlatTaskVisitor();
        t1.accept(txt);
        t2.accept(txt);
        t3.accept(txt);
        txt.complete();

        String out = txt.result();
        Assert.assertTrue(out.contains("Tasks Export"));
        Assert.assertTrue(out.contains("ID: 10"));
        Assert.assertTrue(out.contains("Title: A"));
        Assert.assertTrue(out.contains("State: ToDo"));
        Assert.assertTrue(out.contains("State: InProgress"));
        Assert.assertTrue(out.contains("State: Completed"));
    }

    /**
     * Verifies counting tasks by state via {@code CountByStateVisitor} and
     * checks CSV/plain-text exports of the resulting report.
     */
    @Test
    public void by_state_count_and_exporters() {
        CountByStateVisitor counter = new CountByStateVisitor();

        new Task(21, "T1", "x", TaskState.ToDo).accept(counter);
        new Task(22, "T2", "y", TaskState.ToDo).accept(counter);
        new Task(23, "T3", "z", TaskState.InProgress).accept(counter);
        new Task(24, "T4", "w", TaskState.Completed).accept(counter);
        new Task(25, "T5", "q", TaskState.Completed).accept(counter);

        Report rep = counter.report();
        ByStateCount byState = (ByStateCount) rep;

        Assert.assertEquals(2, byState.count(TaskState.ToDo));
        Assert.assertEquals(1, byState.count(TaskState.InProgress));
        Assert.assertEquals(2, byState.count(TaskState.Completed));

        IReportExporter<ByStateCount> csv = new ByStateCsvExporter();
        String csvOut = csv.export(byState);
        Assert.assertTrue(csvOut.startsWith("state,count"));
        Assert.assertTrue(csvOut.contains("ToDo,2"));
        Assert.assertTrue(csvOut.contains("InProgress,1"));
        Assert.assertTrue(csvOut.contains("Completed,2"));

        IReportExporter<ByStateCount> txt = new ByStatePlainTextExporter();
        String txtOut = txt.export(byState);
        Assert.assertTrue(txtOut.contains("Tasks by state"));
        Assert.assertTrue(txtOut.contains("ToDo: 2"));
        Assert.assertTrue(txtOut.contains("InProgress: 1"));
        Assert.assertTrue(txtOut.contains("Completed: 2"));
    }
}
```

C:\Users\Itay_Vazana\Desktop\BSc CS\Design Patterns\Final_Project\Task_Management_Appliction\src\test\taskmanagement\persistence\TaskDaoTest.java

TaskDaoTest

```java
package taskmanagement.persistence;

import org.junit.*; // JUnit 4
import org.junit.Test;
import org.junit.Before;
import org.junit.AfterClass;
import org.junit.FixMethodOrder;
import org.junit.runners.MethodSorters;

import taskmanagement.domain.ITask;
import taskmanagement.domain.Task;
import taskmanagement.domain.TaskState;
import taskmanagement.persistence.derby.EmbeddedDerbyTasksDAO;

/**
 * JUnit 4 CRUD tests for {@link EmbeddedDerbyTasksDAO}.
 * <p>Execution order is fixed by method name to make assertions deterministic.</p>
 */
@FixMethodOrder(MethodSorters.NAME_ASCENDING) // optional: run by method name
public class TaskDaoTest {

    private EmbeddedDerbyTasksDAO dao;

    /**
     * Initializes the DAO and clears all rows before each test.
     *
     * @throws TasksDAOException if clearing the table fails
     */
    @Before
    public void setUp() throws TasksDAOException {
        dao = EmbeddedDerbyTasksDAO.getInstance();
        dao.deleteTasks();
    }

    /**
     * Shuts down the embedded Derby DAO after all tests complete.
     */
    @AfterClass
    public static void afterAll() {
        EmbeddedDerbyTasksDAO.getInstance().shutdown();
    }

    /**
     * Verifies that adding a task assigns an id and that listing returns
     * the inserted task with expected field values.
     *
     * @throws TasksDAOException if DAO operations fail
     */
    @Test
    public void test01_add_and_list() throws TasksDAOException {
        Task t = new Task(0, "Write tests", "DAO CRUD", TaskState.ToDo);
        dao.addTask(t);

        Assert.assertTrue("Expected DAO to assign id > 0", t.getId() > 0);

        ITask[] all = dao.getTasks();
        Assert.assertEquals(1, all.length);
        Assert.assertEquals("Write tests", all[0].getTitle());
        Assert.assertEquals(TaskState.ToDo, all[0].getState());
```

TaskDaoTest

```java
    }

    /**
     * Verifies retrieval by id and not-found behavior.
     *
     * @throws TasksDAOException if DAO operations fail
     */
    @Test
    public void test02_get_by_id_and_not_found() throws TasksDAOException {
        Task t = new Task(0, "A", "desc", TaskState.InProgress);
        dao.addTask(t);
        int id = t.getId();

        ITask fromDb = dao.getTask(id);
        Assert.assertEquals("A", fromDb.getTitle());
        Assert.assertEquals(TaskState.InProgress, fromDb.getState());

        Assert.assertThrows(TasksDAOException.class, () -> dao.getTask(999_999));
    }

    /**
     * Verifies insertion with an explicit id and duplicate key rejection.
     *
     * @throws TasksDAOException if DAO operations fail unexpectedly
     */
    @Test
    public void test03_add_with_explicit_id_and_duplicate() throws TasksDAOException {
        Task t1 = new Task(42, "X", "first", TaskState.ToDo);
        dao.addTask(t1);
        Assert.assertEquals(42, t1.getId());

        Task t2 = new Task(42, "Y", "dup", TaskState.ToDo);
        Assert.assertThrows(TasksDAOException.class, () -> dao.addTask(t2));
    }

    /**
     * Verifies updating an existing row and failure when updating a missing id.
     *
     * @throws TasksDAOException if DAO operations fail
     */
    @Test
    public void test04_update_existing_and_missing() throws TasksDAOException {
        Task t = new Task(0, "Before", "desc", TaskState.ToDo);
        dao.addTask(t);
        int id = t.getId();

        Task updated = new Task(id, "After", "desc2", TaskState.Completed);
        dao.updateTask(updated);

        ITask fromDb = dao.getTask(id);
        Assert.assertEquals("After", fromDb.getTitle());
        Assert.assertEquals("desc2", fromDb.getDescription());
        Assert.assertEquals(TaskState.Completed, fromDb.getState());

        Task missing = new Task(123456, "Nope", "NA", TaskState.ToDo);
        Assert.assertThrows(TasksDAOException.class, () -> dao.updateTask(missing));
    }

    /**
```

TaskDaoTest

```java
     * Verifies single-row deletion and not-found behavior on repeated deletion.
     *
     * @throws TasksDAOException if DAO operations fail
     */
    @Test
    public void test05_delete_single_and_missing() throws TasksDAOException {
        Task t = new Task(0, "To delete", "d", TaskState.ToDo);
        dao.addTask(t);
        int id = t.getId();

        dao.deleteTask(id);
        Assert.assertThrows(TasksDAOException.class, () -> dao.getTask(id));
        Assert.assertThrows(TasksDAOException.class, () -> dao.deleteTask(id));
    }

    /**
     * Verifies bulk deletion clears the table.
     *
     * @throws TasksDAOException if DAO operations fail
     */
    @Test
    public void test06_delete_all() throws TasksDAOException {
        dao.addTask(new Task(0, "A", "1", TaskState.ToDo));
        dao.addTask(new Task(0, "B", "2", TaskState.Completed));
        Assert.assertTrue(dao.getTasks().length >= 2);

        dao.deleteTasks();
        Assert.assertEquals(0, dao.getTasks().length);
    }
}
```