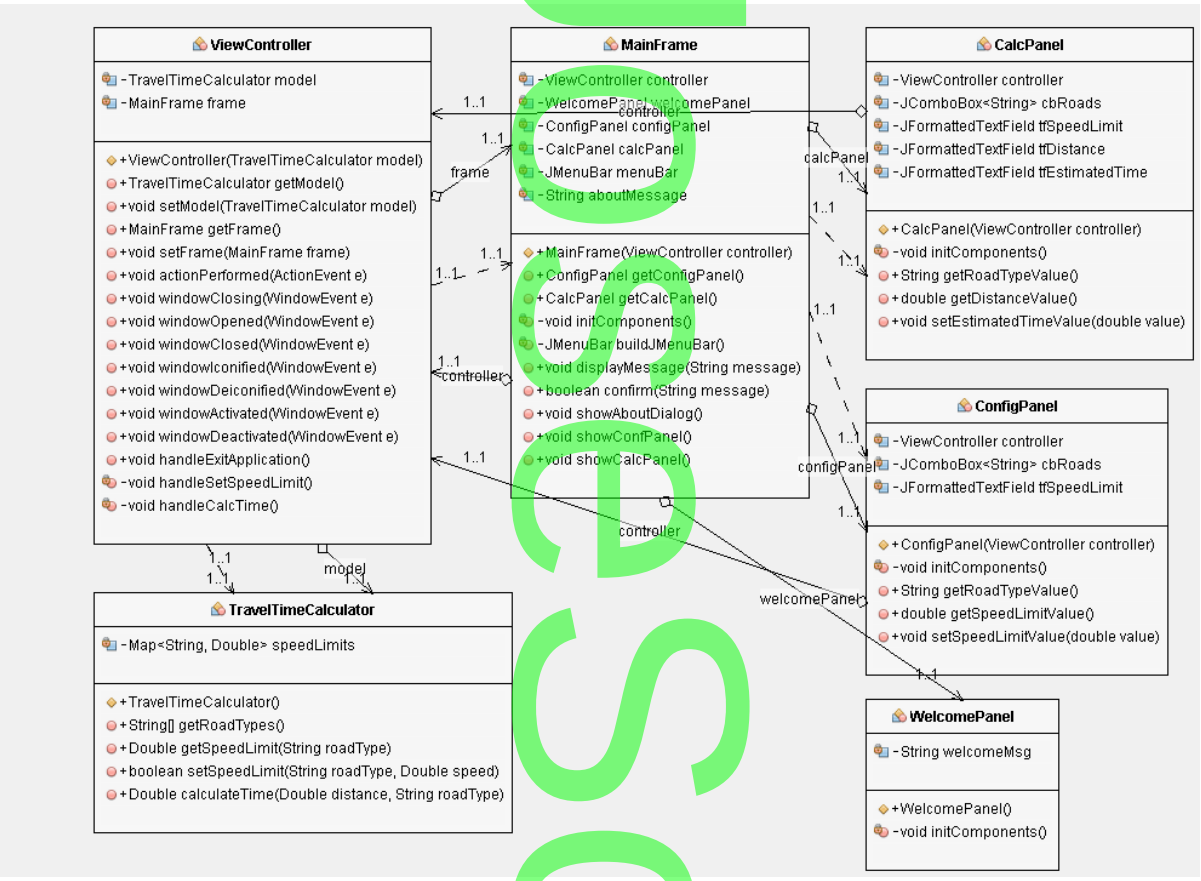
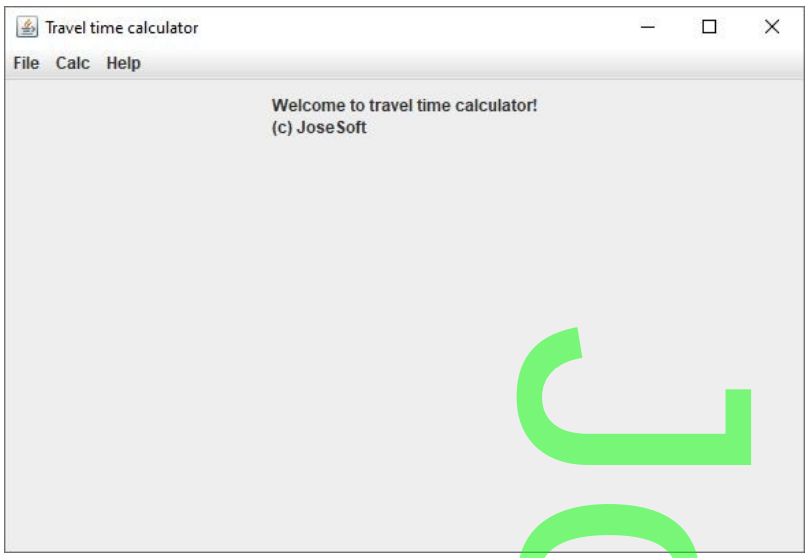
GUI i MVC: Comunicació entre components gràfics, controladors i model

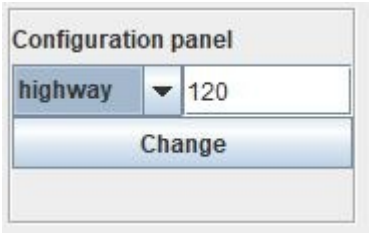
Usarem com a exemple una aplicació gràfica per al càlcul del temps estimat de viatge donada una distància i un tipus de via. Cada tipus de via té associat un límit màxim de velocitat. El càlcul del temps es realitza suposant que es viatja a la velocitat màxima permesa en el tipus de via especificat.

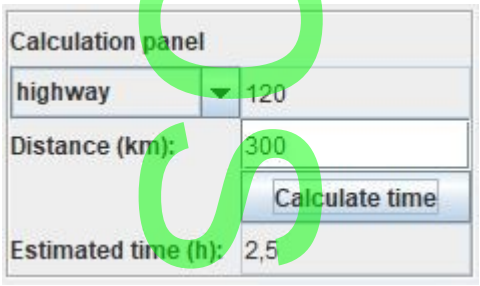


L'aplicació consta d'una finestra principal en la qual es carreguen tres tipus de panells: panell de benvinguda, panell de configuració (per establir les velocitats límit en cada via) i panell de càlcul de temps de viatge.

Existeix un model per emmagatzemar les dades dels tipus de via i els seus límits de velocitat, així com per proveir els càlculs per a l'aplicació.








```java
public class TravelTimeCalculator {
    /**
     * the map from road types to speed limits.
     */
    private final Map<String, Double> speedLimits;

    public TravelTimeCalculator() {
        //initialize values of speed limits and road types.
        this.speedLimits = new HashMap<>();
        this.speedLimits.put("urban", 50.0);
        this.speedLimits.put("motorway", 100.0);
        this.speedLimits.put("highway", 120.0);
    }

    /**
     * gets the names of the road types.
     * @return the types of the roads.
     */
    public String [] getRoadTypes() {
        String [] types =
this.speedLimits.keySet().stream().toArray(String[]::new);
        return types;
    }

    /**
     * gets speed limit of the given road type.
     * @param roadType the type of the road.
     * @return its speed limit, if found, null otherwise,
     */
    public Double getSpeedLimit(String roadType) {
        return this.speedLimits.get(roadType);
    }
```

Jose Soft

```
    /**
     * sets the speed limit of the given road type, only if it exists.
     * @param roadType the type of the road.
     * @param speed the speed of the road.
     * @return true if set, false otherwise.
     */
    public boolean setSpeedLimit(String roadType, Double speed) {
        if (this.getSpeedLimit(roadType)==null) {
            return false;
        }
        this.speedLimits.put(roadType, speed);
        return true;
    }

    /**
     * calculates estimated time of travel.
     * @param distance the distance to travel, in km.
     * @param roadType the type of the road.
     * @return the estiimated time of travelling, in hours.
     */
    public Double calculateTime(Double distance, String roadType) {
        Double speed = this.getSpeedLimit(roadType);
        Double time = distance/speed;
        return time;
    }

}
```

Per iniciar l'aplicació, s'instancia el model, després el controlador, al qual cal passar-li el model com a paràmetre.

```
/**
 * Main class for travel time calculator application.
 * This class is the entry point to the application.
 * @author Jose
 */
public class Main {
    public static void main(String[] args) {
        //instantiate model.
        TravelTimeCalculator ttcalc = new TravelTimeCalculator();
        //instantiate controller.
        ViewController controller = new ViewController(ttcalc);
        javax.swing.SwingUtilities.invokeLater(() -> {
            //instantiate main view.
            MainFrame frame = new MainFrame(controller);
            //assign view to controller.
            controller.setFrame(frame);
            //make view visible.
            frame.setVisible(true);
        });
    }
}
```

Després es crea la vista principal, la qual ha de tenir accés al controlador i al model. Podem donar-li aquest accés de diverses maneres. A tall d'exemple, les següents:

- Passar-li al constructor el controlador i el model.
- Passar-li al constructor el controlador i incloure al controlador un getter

per al model.

L'opció exemplificada aquí és la segona.

```
public class ViewController implements ActionListener, WindowListener {

    /**
     * the model to manage data.
     */
    private TravelTimeCalculator model;
    /**
     * the main view.
     */
    private MainFrame frame;

    public ViewController(TravelTimeCalculator model) {
        this.model = model;
    }

    public TravelTimeCalculator getModel() {
        return model;
    }

    public void setModel(TravelTimeCalculator model) {
        this.model = model;
    }

    public MainFrame getFrame() {
        return frame;
    }

    public void setFrame(MainFrame frame) {
        this.frame = frame;
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String action = e.getActionCommand();
        if (action != null) {
            switch (action) {
                case "exit":
                    handleExitApplication();
                    break;
                case "about":
                    frame.showAboutDialog();
                    break;
                case "confpanel":
                    frame.showConfPanel();
                    break;
                case "calcform":
                    frame.showCalcPanel();
                    break;
                case "set_speed":
                    handleSetSpeedLimit();
                    break;
                case "calc_time":
                    handleCalcTime();
                    break;
                default:
                    frame.displayMessage(String.format("Action %s", action));
                    break;
```

Tipsos

```
            }
        }
    }

    /* handlers of window events */

    @Override
    public void windowClosing(WindowEvent e) {
        handleExitApplication();
    }

    /**
     * handle method to exit application.
     */
    public void handleExitApplication() {
        if (frame.confirm("Exit application. Are you sure")) {
            System.exit(0);
        }
    }

    /**
     * handle method to set speed limit.
    */
    private void handleSetSpeedLimit() {
        String roadType = getFrame().getConfigPanel().getRoadTypeValue();
        double speed = getFrame().getConfigPanel().getSpeedLimitValue();
        boolean result = model.setSpeedLimit(roadType, speed);
        if (result) {
            JOptionPane.showMessageDialog(frame, "Successfully changed");
        } else {
            JOptionPane.showMessageDialog(frame, "Speed not changed");
        }
    }

    /**
     * handle method to calculate estimated time.
    */
    private void handleCalcTime() {
        double distance = getFrame().getCalcPanel().getDistanceValue();
        String roadType = getFrame().getCalcPanel().getRoadTypeValue();
        double estimatedTime = model.calculateTime(distance, roadType);
        getFrame().getCalcPanel().setEstimatedTimeValue(estimatedTime);
    }

}
```

Per tractar els esdeveniments de la interfície gràfica cal crear almenys un *Listener*. Podem crear un listener per a cada component gràfic (la finestra principal i els diversos panells) o un únic listener per a tots. En aquest exemple, s'ha optat per un únic listener per a tots.

```
public class MainFrame extends JFrame {

    private final ViewController controller;
    //
    private WelcomePanel welcomePanel;
    private ConfigPanel configPanel;
    private CalcPanel calcPanel;
    private JMenuBar menuBar;
    private final String aboutMessage;
```

```
    public MainFrame(ViewController controller) {
        this.controller = controller;
        initComponents();
        aboutMessage = String.format(
                "<html><p><strong>%s</strong></p><p>(c)ProvenSoft</p></html>",
                getTitle());
    }

    public ConfigPanel getConfigPanel() { ... }

    public CalcPanel getCalcPanel() { ... }

    private void initComponents() { ... }

    private JMenuBar buildJMenuBar() { ... }

    public void displayMessage(String message) { ... }

    public boolean confirm(String message) { ... }

    public void showAboutDialog() {
        JOptionPane.showMessageDialog(this, aboutMessage);
    }

    public void showConfPanel() {
        Container pane = getContentPane();
        pane.removeAll();
        repaint();
        configPanel = new ConfigPanel(controller);
        pane.setLayout(new FlowLayout());
        pane.add(configPanel);
        this.validate();
    }

    public void showCalcPanel() {
        Container pane = getContentPane();
        pane.removeAll();
        repaint();
        calcPanel = new CalcPanel(controller);
        pane.setLayout(new FlowLayout());
        pane.add(calcPanel);
        this.validate();
    }

}
```

Els panells han de tenir accés també al controlador i al model, així com a un *listener* per als seus esdeveniments.

Entre les moltes opcions possibles, podem citar les següents:

- *MainFrame* és *listener* de sí mateix i de tots els panells. Cal passar als panells *MainFrame* i el model (si no té accés a través de *MainFrame*).
- Fem que el controlador sigui *listener*. Passem el controlador als panells. Com en el cas anterior, cal també donar als panells accés al model (això es pot aconseguir passant-lo com a paràmetre al constructor, o generant un getter al controlador).

L'opció escollida és la segona.

OSSOL

```java
public class ConfigPanel extends JPanel {

    private final ViewController controller;
    private JComboBox<String> cbRoads;
    private JFormattedTextField tfSpeedLimit;

    public ConfigPanel(ViewController controller) {
        this.controller = controller;
        initComponents();
    }

    private void initComponents() { ... }

    public String getRoadTypeValue() {
        return (String) cbRoads.getSelectedItem();
    }

    public double getSpeedLimitValue() {
        return (double) tfSpeedLimit.getValue();
    }

    public void setSpeedLimitValue(double value) {
        tfSpeedLimit.setValue(value);
    }

}
```

```java
public class CalcPanel extends JPanel {

    private final ViewController controller;
    private JComboBox<String> cbRoads;
    private JFormattedTextField tfSpeedLimit;
    private JFormattedTextField tfDistance;
    private JFormattedTextField tfEstimatedTime;

    public CalcPanel(ViewController controller) {
        this.controller = controller;
        initComponents();
    }

    private void initComponents() { ... }

    public String getRoadTypeValue() {
        return (String) cbRoads.getSelectedItem();
    }

    public double getDistanceValue() {
        return (double) tfDistance.getValue();
    }

    public void setEstimatedTimeValue(double value) {
        tfEstimatedTime.setValue(value);
    }

}
```

Tosdaf