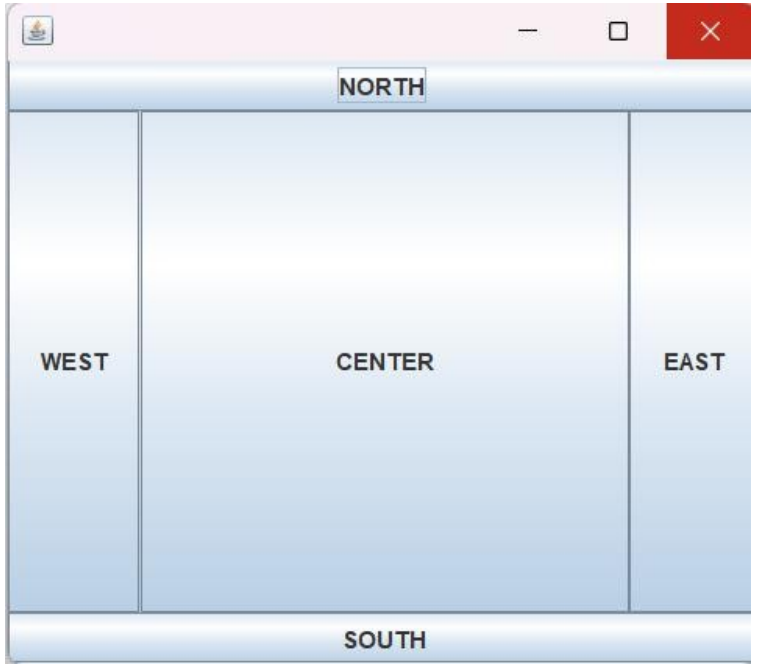
Object-Oriented Programming

6. Lecture
LS 2025/2026
Author: Juraj Petrík

Layout

- Adding widgets (components) to the frame
- Drawing 2D graphics
- Inserting an image



- https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html

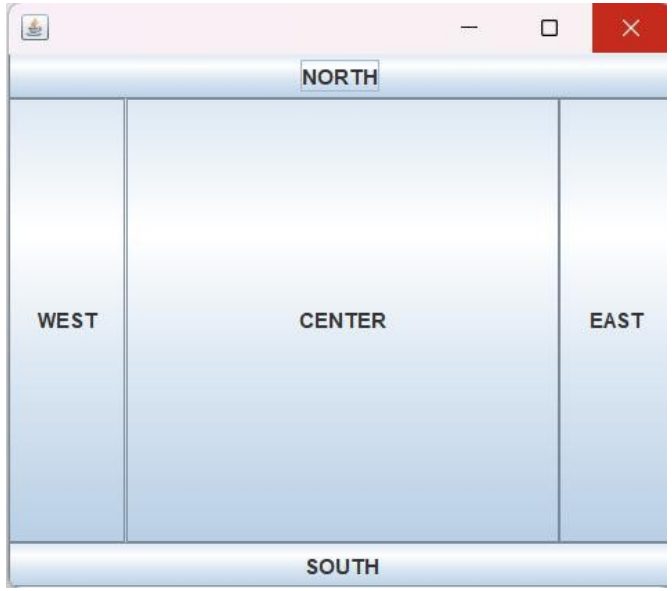
Layout managers

- Components can be nested within each other as standard

- BorderLayout – one component per region, usually does not respect component dimensions (default for frame)
- FlowLayout – from left to right, respects component dimensions, wrapping (default for panel)
- BoxLayout – similar to FlowLayout but from top to bottom, does not automatically wrap components
- GridBagLayout -

Layouts

- Note other specifics, this is the "default" behavior



BorderLayout

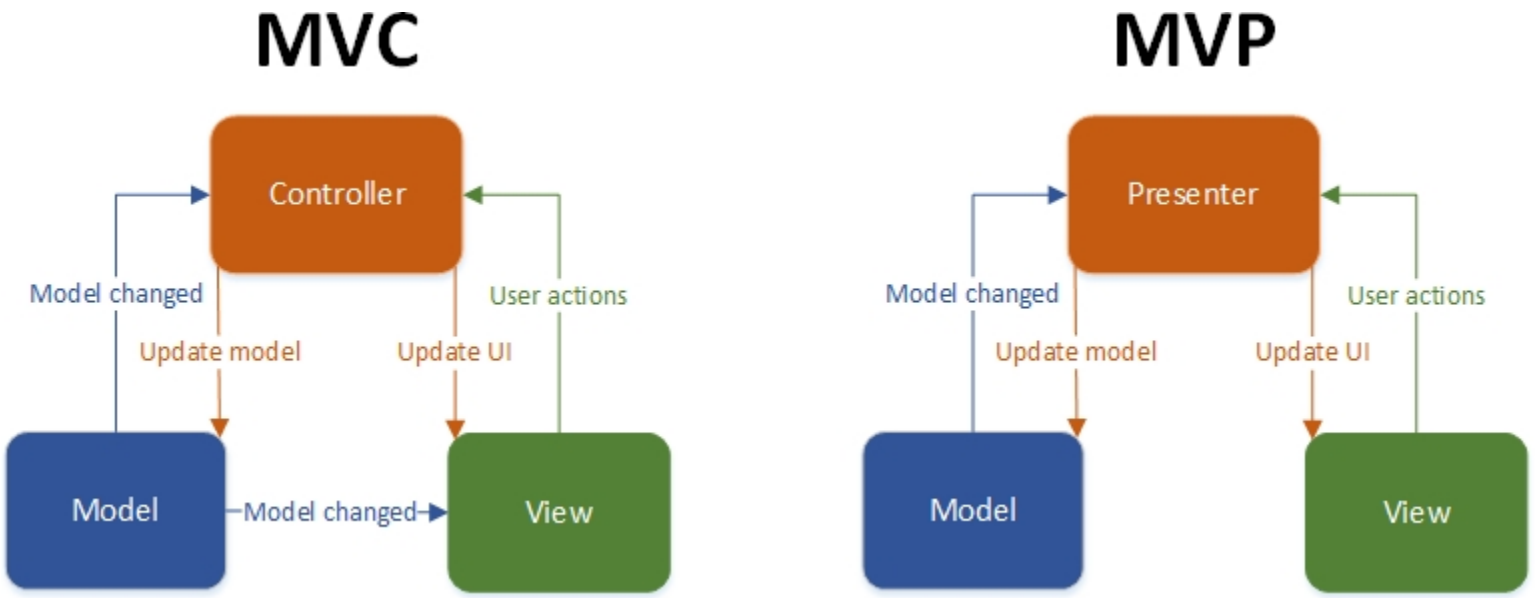
- How do regions behave?
- 1 component per region
- What about the size of components?
- It depends.

Components in Swing

- Composite
- Background (JFrame, JPanel)
- Interactive components:
  - JButton
  - JLabel
  - JTextField
  - JTextArea
  - JList

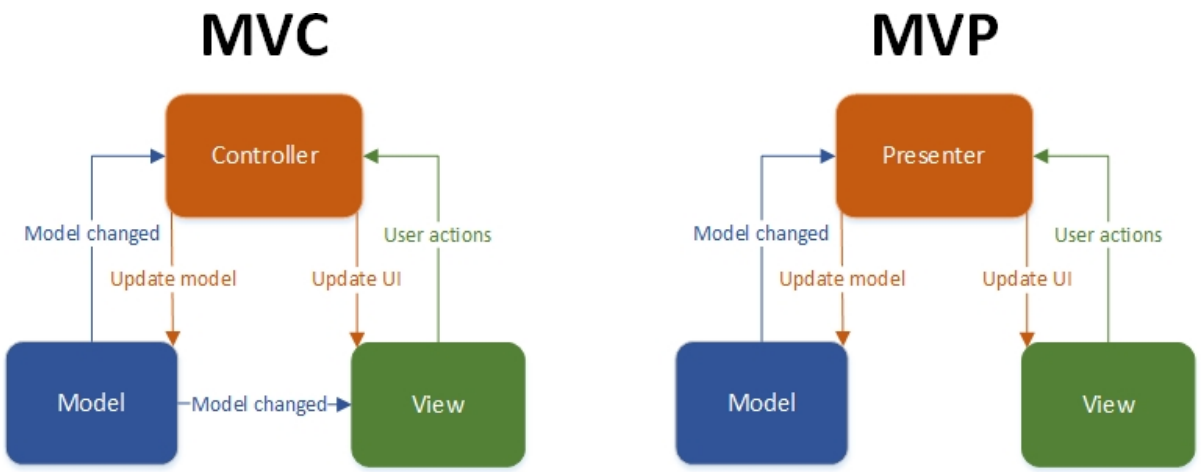
GUI in general

- What happens when something takes longer?
- Separate data, application logic, and presentation layer
- Use an appropriate pattern: MVC vs MVP vs MVVM vs ...



MVC

- The model represents the data model itself
- View handles the GUI
- Controller handles "communication" between the model and view:
  - Implements listeners
  - Validates inputs
  - Updates the model



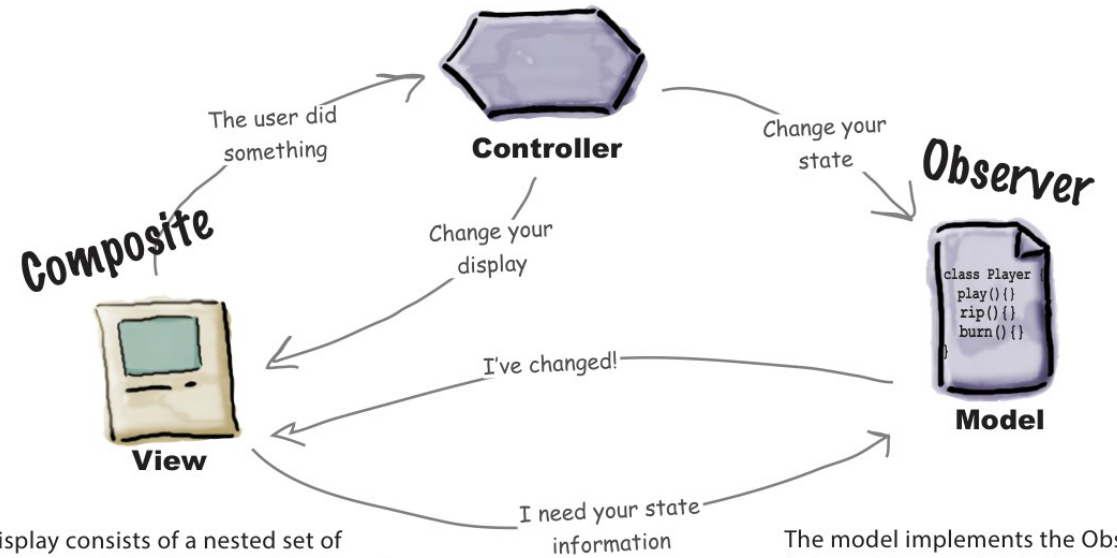
MVC

- View – Composite
- Controller – Strategy
- Model – Observer

- Head First Design Patterns 2nd edition

Strategy

The view and controller implement the classic Strategy Pattern: the view is an object that is configured with a strategy. The controller provides the strategy. The view is concerned only with the visual aspects of the application, and delegates to the controller any decisions about the interface behavior. Using the Strategy Pattern also keeps the view decoupled from the model because it is the controller that is responsible for interacting with the model to carry out user requests. The view knows nothing about how this gets done.

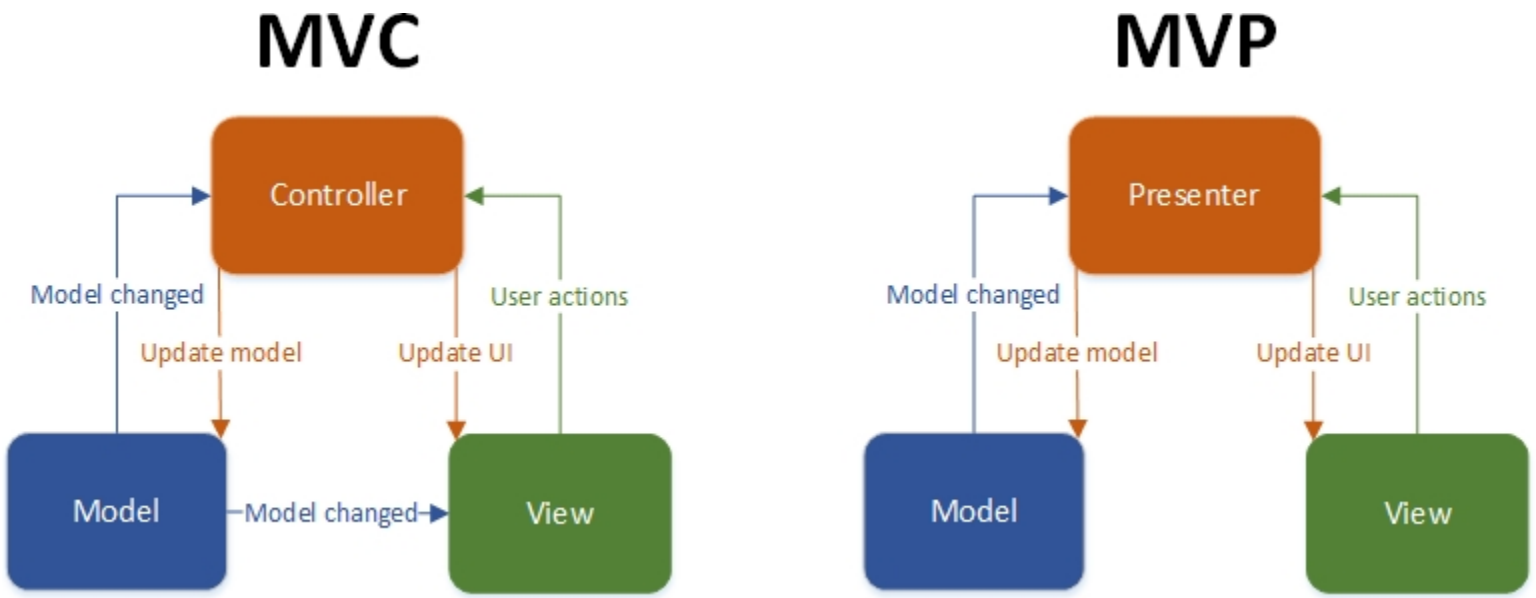


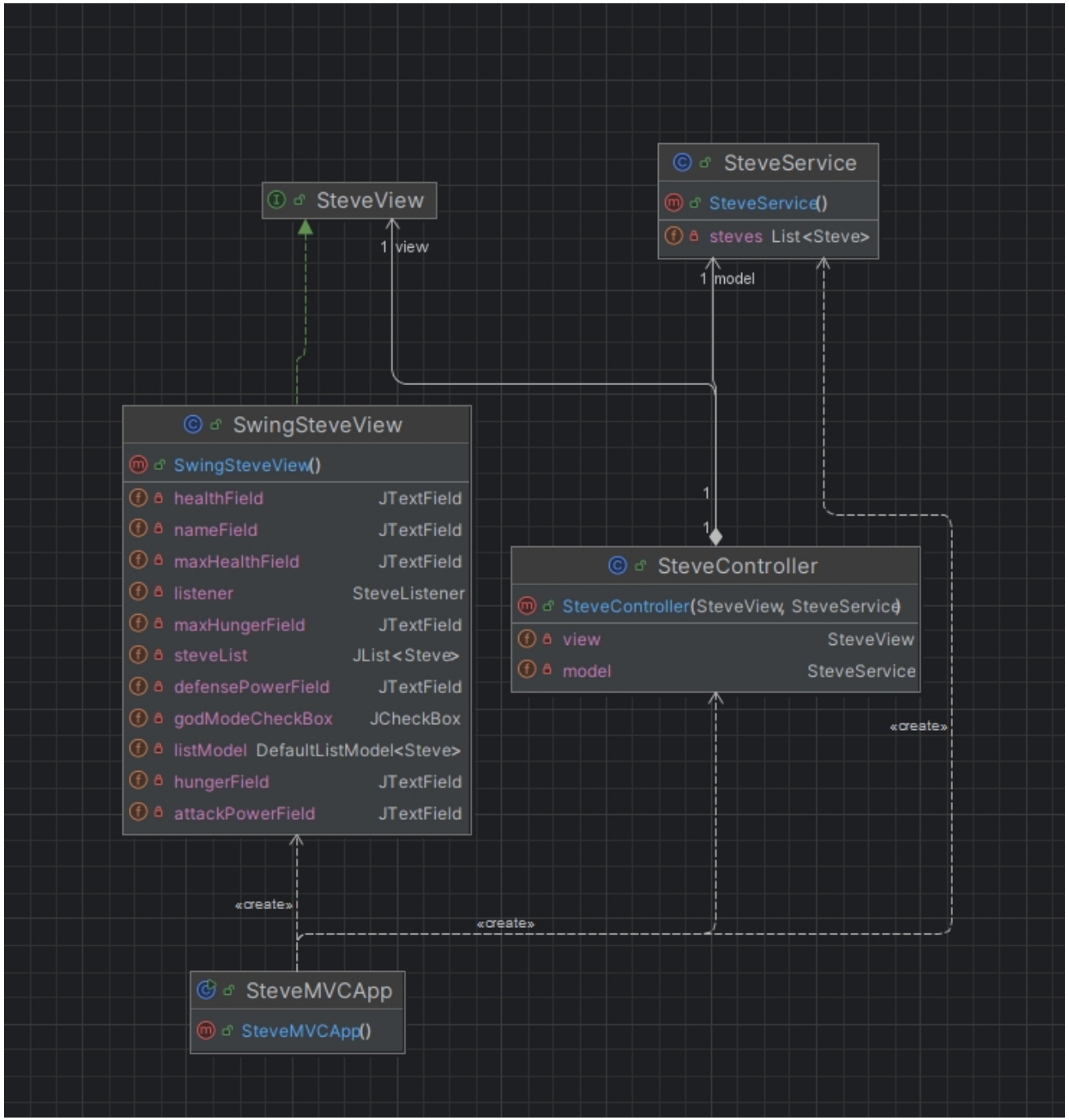
The display consists of a nested set of windows, panels, buttons, text labels, and so on. Each display component is a composite (like a window) or a leaf (like a button). When the controller tells the view to update, it only has to tell the top view component, and Composite takes care of the rest.

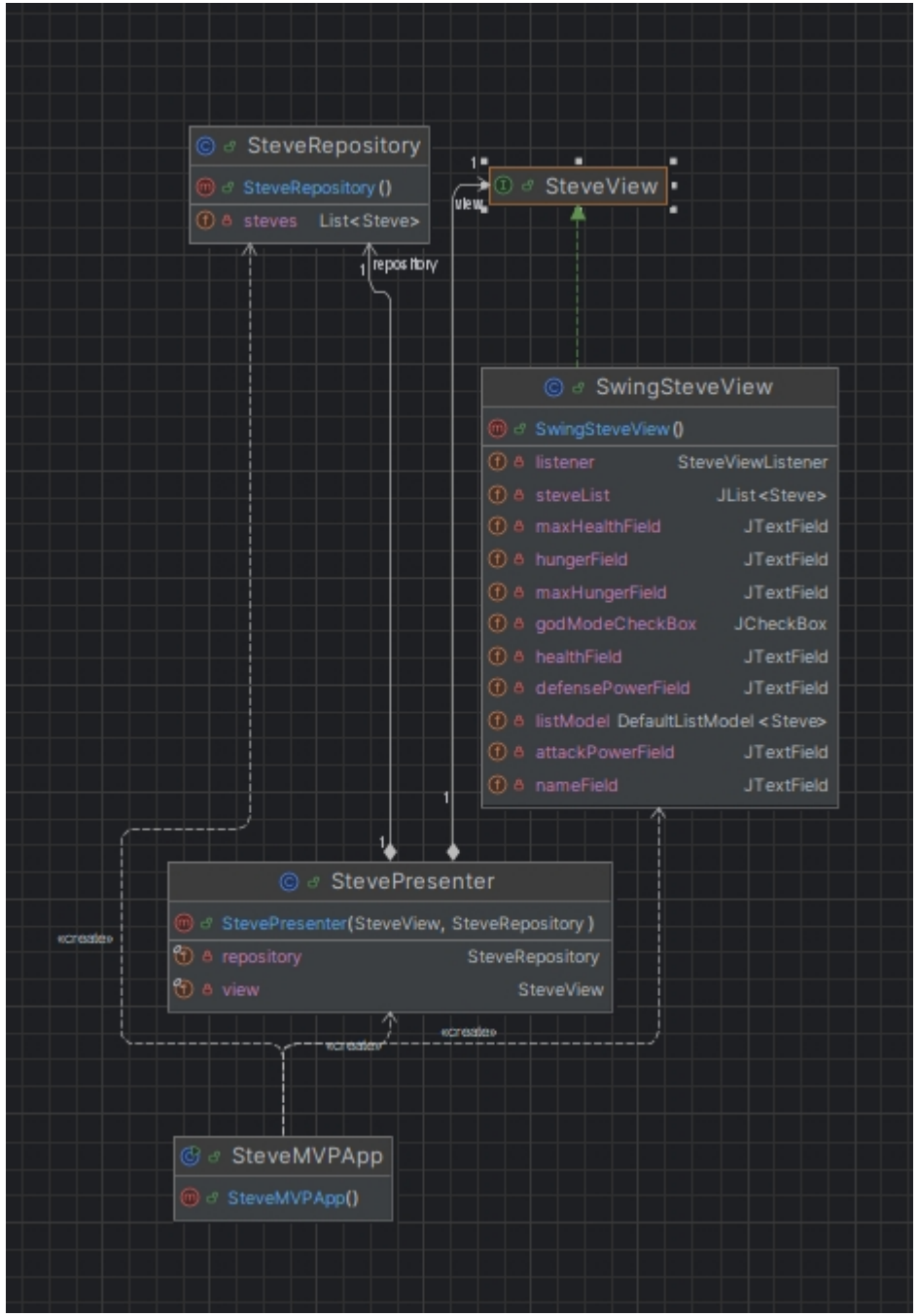
The model implements the Observer Pattern to keep interested objects updated when state changes occur. Using the Observer Pattern keeps the model completely independent of the views and controllers. It allows us to use different views with the same model, or even use multiple views at once.

MVP

- The model represents the data model itself
- View handles the GUI, it is unaware of the models
- Presenter is a mediator between the model and view

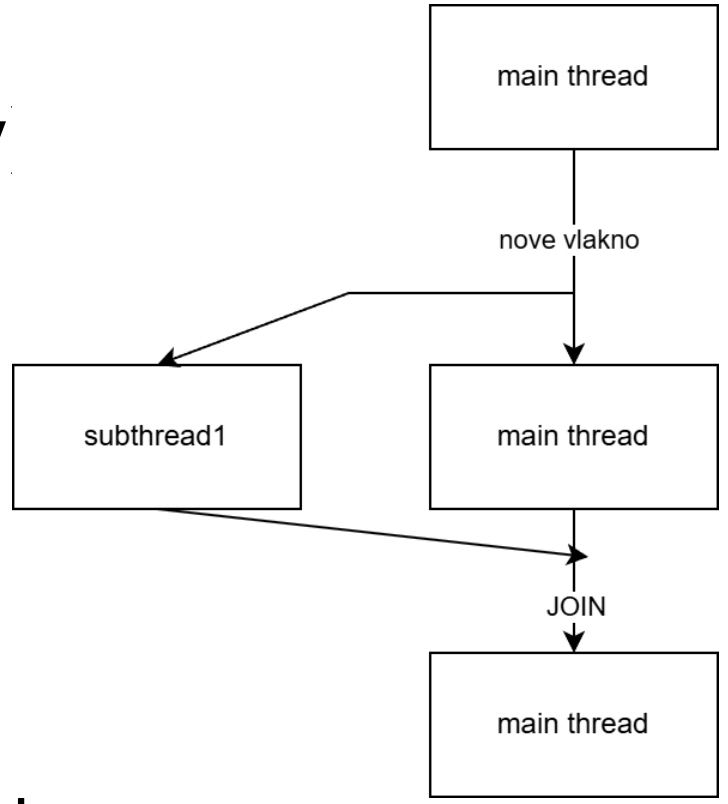



SteveView
SteveService
SteveService()
steves List<Steve>
1 view
1 model
SwingSteveView
SwingSteveView()
healthField JTextField
nameField JTextField
maxHealthField JTextField
listener SteveListener
maxHungerField JTextField
steveList JList<Steve>
defensePowerField JTextField
godModeCheckBox JCheckBox
listModel DefaultListModel<Steve>
hungerField JTextField
attackPowerField JTextField
1
1
SteveController
SteveController(SteveView, SteveService)
view SteveView
model SteveService
«create»
«create»
«create»
SteveMVCApp
SteveMVCApp()


SteveRepository
SteveRepository()
steves List<Steve>
1
view
SteveView
1
repository
SwingSteveView
SwingSteveView()
listener SteveViewListener
steveList JList<Steve>
maxHealthField JTextField
hungerField JTextField
maxHungerField JTextField
godModeCheckBox JCheckBox
healthField JTextField
defensePowerField JTextField
listModel DefaultListModel<Steve>
attackPowerField JTextField
nameField JTextField
1
1
StevePresenter
StevePresenter(SteveView, SteveRepository)
repository SteveRepository
view SteveView
«create»
«create»
«create»
SteveMVPApp
SteveMVPApp()

Parallel processing using threads

- We do two or more things at once (concurrently
- Threads:
  - "Lightweight processes"
  - **Shared memory**
- Beware of determinism



- Implementing Runnable
- Swing, e.g., javax.swing.SwingUtilities.invokeLater

Atomicity

- Synchronized methods
- Only one thread may work with an object at a time
- Each object has a lock, but they are only used if we have synchronized parts of the code
- Synchronized for everything?
  - Deadlock
  - We do not use concurrency
- Atomic variables (AtomicInteger) – incrementAndGet, compareAndSet

Immutable objects

- It is not desirable for objects to change "under the hood"
- Final class
- Final attributes
- Initialize once
- No setters!

- Be careful with collections, use an appropriate class (e.g., CopyOnWriteArrayList) – each process works on a snapshot

Runnable vs callable

- Runnable does not return values (return) and cannot throw checked exceptions

Quiz time

Storing objects

- Saving object state
- Using:
  - Serialization:□□ sr minecraft.player.SteveJHu□□h Z godModeI hungerI maxHungerx
  - Plain Text: Juraj,1,1,1..

Streams

Serialization

- The entire object graph is saved automatically
- All or nothing
- Implements Serializable
- If I don't want to/don't know how to serialize - transient

Deserialization

- The constructor is no longer executed

What if the class changes?

io, nio, nio2