QUICK RECAP

A reminder of what we have done during Lecture 02

● ● ● ●

LAST TIME...

- Component hierarchy
  - What are the basic interfaces?
  - What are the differences?
- Common properties
  - What are they?
  - Where are they defined?
- Components
  - How are they grouped?
  - What events do they broadcast and when?
- Coding
  - How to get rid of widgetset compilation ?

VAADIN FRAMEWORK: EVENTS AND DATA BINDING*

*SUBJECT TO CHANGE IN THE UPCOMING VAADIN 8

Development of Modern Web Applications (with Vaadin)

Lecture 03

OVERVIEW

- Events
- Notifications and windows
- The data model

- Shoutbox app continues

EVENTS AND LISTENERS

(Somewhat) In-depth overview

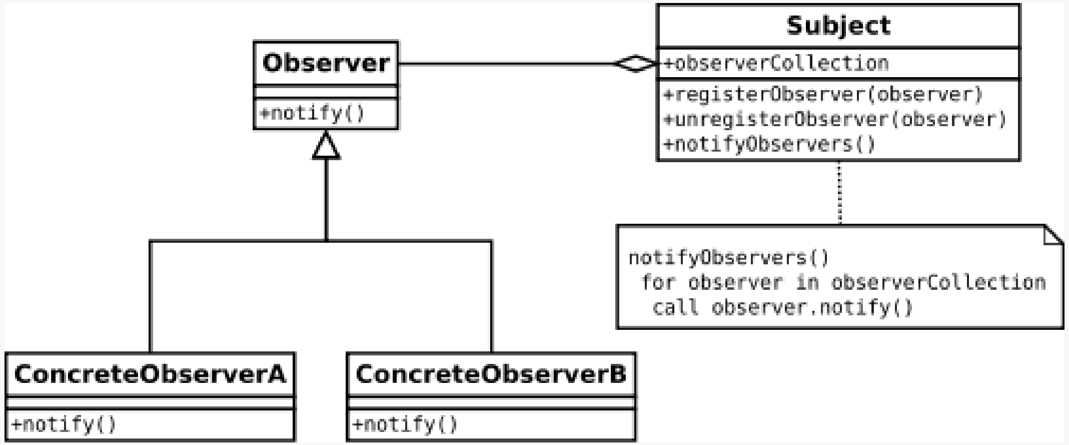
● ● ● ●

THE OBSERVER PATTERN

A subject:
•maintains a list of observers;
•notifies each observer about state changes;
•allows registering new observers and unregistering existing ones.

This approach helps in implementing distributed event handling and is commonly found in GUI toolkits.

Source: wiki
Mandatory read: *Design Patterns*, GOF



```
notifyObservers()
 for observer in observerCollection
  call observer.notify()
```

EVENTS

- Component-specific
  - Button → click
  - Grid → selection event
  - Table → column reorder, resize, click…
  - Tree → collapse, expand
  - TextField → text change listener…
- Group-specific
  - Fields → value changed
  - Selects → underlying data source changed
- Custom
  - Create interfaces and event class
  - Implement notifier interface in your component

THE VAADIN WAY

```
public interface FooNotifier {
  void addFooListener(FooListener foo);
  void removeFooListener(FooListener foo);
} // convention: explicit event broadcasting

public interface FooListener {
  void fooHappened(FooEvent e);
}

public class FooEvent
(extends com.vaadin.ui.Component.Event) {
  //  some methods
} // (built-in support for broadcasting)
  // protected void fireEvent(EventObject e)
  // void addListener(...) - non-deprecated ones
```

HANDLING EVENTS

- Make the class a listener
  - The listener method is part of the class
    - Problems when changing implementation
- Create a listener class
  - Allows reusing listeners
  - *Entia non sunt multiplicanda praeter necessitatem*
    - Entities must not be multiplied beyond necessity
- Use anonymous listener class
  - Private (static final) variable
    - Class-wide reuse
  - Inline
    - No reuse, one-time only
  - Both ways decrease clarity of the code

UM, NO
AND NO.
quickmeme.com

HANDLING EVENTS – JAVA 8

- Method reference
  - `addFooListener(this::onFooEvent)`
  - `private void onFooEvent(FooEvent foo)`
  - Clean, obvious and readable
    - If you follow the naming pattern, that is
- Lambda
  - `addFooListener(e -> ...do things...)`
  - Several use cases, e.g.
    - Pre-processing parameters before handling
    - Choosing different handlers based on event
- Corner cases
  - Just use the previously mentioned approaches

NOTIFICATIONS

Notifying users about events (and other things as well)

com.vaadin.ui.Notification

Purpose

- Notification message popup
- Compact message
  - Priority (type) can vary
  - Not too bloated
  - Draws attention

Properties

- Caption
  - Short, descriptive text
- Description
  - More elaborate text
  - Not too bloated
- Icon
- Position
- Display delay
  - Or close-on-click
- Type
  - Four predefined types

com.vaadin.ui.Notification.Type

HUMANIZED_MESSAGE

- Fades away quickly
  - Mouse moved
  - Keyboard events
- Fairly unimportant messages
  - Dull style by default
  - Can be ignored

WARNING_MESSAGE

- Stays a little after mouse or keyboard events
- Messages that should be noticed
  - But are not critical
  - Style is more visible

TRAY_NOTIFICATION

- Shown in a corner of a browser window
- Notification message
  - Should not interfere with whatever the user is doing

ERROR_MESSAGE

- Must be closed by the user
  - Messages that must be noticed
  - Any critical information
- Very visible style

Showing notifications

Predefined methods

- Static methods in `Notification`
  - Displayed inside the current page
  - Suitable most of the time
- `show(String);`
  - Displays humanised message
  - Meaning: disappears fast
- `show(String, Type);`
  - Caption and type
- `show(String, String, Type);`
  - Caption, description and type
- No helper with an icon ☹

Custom notifications

- `ntf = new Notification();`
  - Requires a page to display the notification in
    - `p = UI.getCurrent().getPage();`
    - `p = Page.getCurrent();`
- `ntf.show(p);`
  - Full control over how the notification is displayed

vaadin}>

SUBWINDOWS

Displaying popups

COM.VAADIN.UI.WINDOW

- Floating panel within a browser window
  - Cannot exist outside the browser
- Single-component container
  - Subclass of `com.vaadin.ui.Panel`
  - So, all the features of a panel
- Attached to UI directly
  - `addWindow(Window window);`
    - Adding twice throws an exception
  - `removeWindow(Window window);`

Properties

Sizeable

- No support for minimising
  - Only maximise and restore
- Can be set programmaticaly

Moveable

- Restricted to browser window
- Can be set programmaticaly

Closeable

- True by default
  - Broadcasts events
- Can be achieved programmaticaly
  - `UI#removeWindow(Window w);`
  - `window.close();`

Modal

- Steals focus
  - Must be closed to continue
- Only the most recent is focused
- Not modal by default
- Browser-side feature

```
Window win =
         new Window("Hello!");
win.setModal(true);

win.setContent(
  new Image("O hai!",
  new ExternalResource(
  "http://bit.ly/10vNQk"
)));

// attaching to UI
UI.getCurrent().
               addWindow(win);
```

COM.VAADIN.UI.WINDOW

CSS rules:
- .v-window
  - .v-caption
  - .v-window-content

DEMO!

Shoutbox step 3
http://github.com/vaadin-miki/shoutbox

end branch: `step-03`

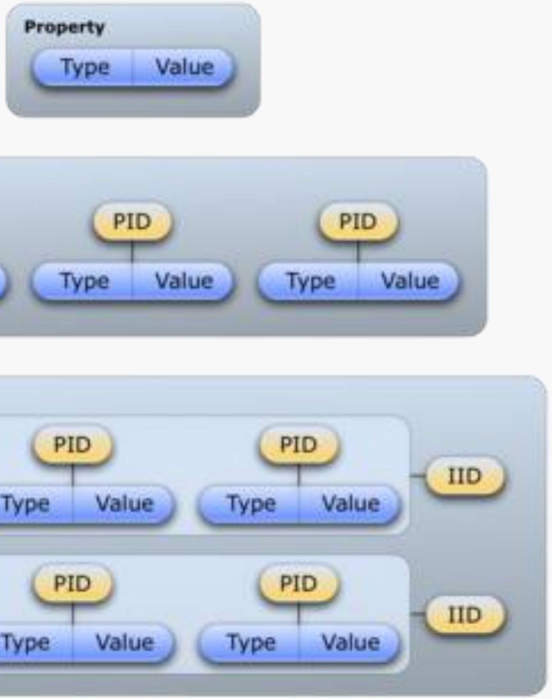
THE PLAN

- Add error notification for entering empty text
- Add a filter for seven dirty words
  - Plus one extra for testing
  - Java resource
- Clean the field after submission
- Enter/Return key submits

THE VAADIN DATA MODEL*

* Subject to change in Vaadin 8

THE VAADIN 7 DATA MODEL



KEY PARTS

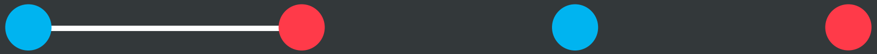
- Property
    - Typed value
- Item
    - Set of uniquely identified properties
- Container
    - Set of uniquely identified items

KEY CHARACTERISTICS

- Interfaces
  - There is no implementation in the model
    - There is implementation in the framework
  - Any data component can use any implementation
  - Underlying data is transparent
- Events
  - Built-in support for events
    - The Vaadin way = explicit event broadcasting
- Type safety
  - Generics
  - Validators
- Extensions
  - New data source = new container

PROPERTY

Typed value



COM.VAADIN.DATA.PROPERTY<T>

- Value → `<T>`
  - Optional events on value change
    - `Property.ValueChangeNotifier`
- Type → `Class<? extends T>`
  - Cannot be modified
  - Can be used to typecast value
    - `getType().cast(getValue);`
- Can be read-only
  - Optional events on status change
  - `Property.ReadOnlyStatusChangeNotifier`
- Implemented by almost all GUI components

COM.VAADIN.DATA. PROPERTY.TRANSACTIONAL<T>

- Provides support for buffering
  - Restoring previous property value
  - Not necessarily database transaction

```
extends Propery<T> {
  void startTransaction();
  void commit();
  void rollback();
}
```

COM.VAADIN.DATA.
PROPERTY.VIEWER

- Interface for components
  - Showing value from a property
  - Can modify it, though
    - Purely informational purpose
- No generics
  - Typecasting is needed

```
getPropertyDataSource() → Property;
setPropertyDataSource(Property);
```

COM.VAADIN.DATA.PROPERTY.EDITOR

- Interface for components
  - Showing and editing value from a property
  - Same methods as viewer
    - Purely informational purpose
- No generics
  - Typecasting is needed

```
getPropertyDataSource() → Property;
setPropertyDataSource(Property);
```

ITEM

Set of uniquely identified properties

COM.VAADIN.DATA.ITEM

- Set of uniquely identified properties
  - Identifier can be any object
  - `getItemPropertyIds() → Collection<?>`
    - Use order-preserving collection
- Adding and removing properties
  - Optional
  - `Item.PropertySetChangeNotifier`
- No generics
  - Typecasting needed

COM.VAADIN.DATA.<u>ITEM.VIEWER</u>
COM.VAADIN.DATA.<u>ITEM.EDITOR</u>

- Interfaces for components
  - Viewing and writing contents of an item
  - No difference
- Methods present in `FieldGroup`
  - No idea why the interface is <u>still</u> not there

```
getItemDataSource() → Item;
setItemDataSource(Item);
```

CONTAINER

Collection of items

COM.VAADIN.DATA.CONTAINER

- Set of uniquely identified items
  - No particular item order
  - No duplicate item identifiers
  - Identifier can be any object
- Item constraints
  - The same number of properties
  - The same property identifiers
  - The same property types
  - Non-null item identifiers
- Lots of optional functionality

com.vaadin.data.Container

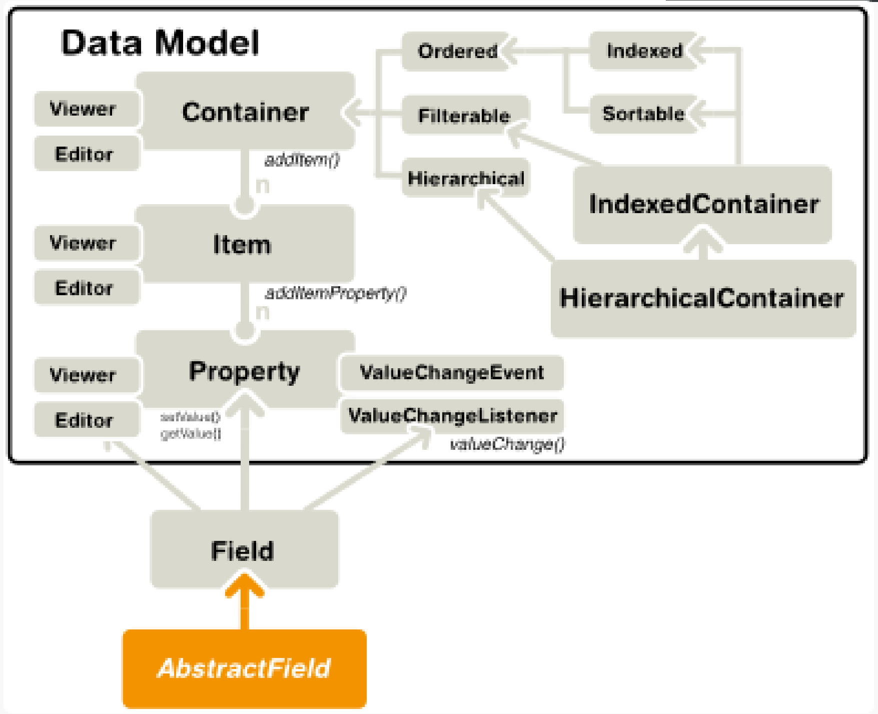
Items

- Obtaining and querying
  - `size() → int;`
  - `getItemIds() → Collection<?>;`
    - This can potentially be time consuming
  - `getItem(Object) → Item;`
  - `containsId(Object) → boolean;`
- Adding or removing
  - Optional
  - `Container.ItemSetChangeNotifier`

Properties

- Obtaining and querying
  - `getContainerPropertyIds() → Collection<?>;`
  - `getContainerProperty( Object, Object) → Property<?>;`
    - Item id, property id
  - `getType(Object) → Class<?>;`
- Adding or removing
  - Optional
  - `Container. PropertySetChangeNotifier`

COM.VAADIN.DATA.CONTAINER



COM.VAADIN.DATA.CONTAINER.VIEWER
COM.VAADIN.DATA.CONTAINER.EDITOR

- Interfaces for components
  - Viewing and writing contents of a container
  - No difference
- All selects are container viewers
  - Options to chose from
  - Some support adding new items
  - All refresh when data source changes

```
getContainerDataSource() → Container;
setContainerDataSource(Container);
```

COM.VAADIN.DATA. CONTAINER.ORDERED

- Provides order of items
  - `firstItemId() → Object;`
  - `isFirstId(Object) → boolean;`
  - `lastItemId() → Object;`
  - `isLastId(Object) → boolean;`
  - `nextItemId(Object) → Object;`
  - `prevItemId(Object) → Object;`
- Inserting items after other items
  - Optional

COM.VAADIN.DATA. CONTAINER.INDEXED

- More than ordered
- Provides non-negative integer index
  - May differ from the item identifier
    - Probably will
  - May change for each item
    - Likely will
  - `getIdByIndex(int index) → Object;`
  - `indexOfId(Object) → int;`
  - `getItemIds(int start, int count) → List<?>;`
- Inserting items at specified position
  - Optional

COM.VAADIN.DATA.
CONTAINER.SORTABLE

- More than ordered
- Allows sorting items
  - Affects their order
  - Affects indices (if container is indexed at the same time)
  - `sort(Object[], boolean[]);`
    - Properties, ascending (default; false = descending)
    - In-place sorting (probably should broadcast an event)
- Sortable properties
  - `getSortableContainerPropertyIds()` → `Collection<?>`
- Adding items
  - Optional
  - Complicated
    - `addItemAfter / addItemAt` may move the item due to sorting

COM.VAADIN.DATA.CONTAINER.FILTERABLE

- Reduces visible items
  - Custom filters
    - Bunch of built-in ones
    - `interface Container.Filter`
  - In-place filtering
    - Should probably broadcast event
  - Affects order and indices
  - Affected by sorting
- Adding items
  - Optional
  - Complicated

COM.VAADIN.DATA. CONTAINER.HIERARCHICAL

- Parent-child relation between items
  - `getChildren(Object) → Collection<?>;`
  - `hasChildren(Object) → boolean;`
  - `getParent(Object) → Object;`
- Many root elements
  - `rootItemIds() → Collection<?>;`
  - `isRoot(Object) → boolean;`
- Explicit ability or disability to have sub-items
- Moving items
  - Optional
  - Should probably broadcast an event
- Sorting, ordering and indexing is complicated
  - Implementation specific

DATA MODEL UTILITIES

Built-in useful classes

COM.VAADIN.DATA.UTIL. OBJECTPROPERTY<T>

- Straightforward implementation
- Broadcasts events

```
new ObjectProperty(T);            // T != null
new ObjectProperty(T, Class<T>);

// with read-only flag
new ObjectProperty(T, Class<T>, boolean);
```

COM.VAADIN.DATA.UTIL. METHODPROPERTY<T>

- Binds property to setter/getter pair
- Useful with beans
  - Any object
  - Any method
    - Well, almost

```
// type, instance, getter, setter
new MethodProperty(Class<? extends T>,
                   Object, Method, Method);
new MethodProperty(Class<? extends T>,
                   Object, String, String);
// more constructors to support parameters
```

COM.VAADIN.DATA.UTIL.ABSTRACTPROPERTY<T>

- Abstract class
- Handles listeners
- Defines methods for firing events
  - Does not fire events

COM.VAADIN.DATA.UTIL. BEANITEM<BT>

- Makes any class an item
  - Requires setters and getters
- Generic bean type
  - `getBean() → BT;`

```
// all bean properties
new BeanItem(BT);
// subset of properties
new BeanItem(BT, Collection<?>);
new BeanItem(BT, String[]);
```

COM.VAADIN.DATA.UTIL.BEANITEMCONTAINER<BT>

- In-memory container for beans
  - All changes are lost
- Uses bean items
  - Generic bean type
  - Beans as identifiers
    - `getItemIds() → List<BT>;     // gives beans`
    - `getItem(BT) → BeanItem<BT>; // gives items`
  - Requires meaningful `hashCode()` in bean
- Filterable, indexed, ordered and sortable
  - And broadcasts events
- No support for adding or removing properties

COM.VAADIN.DATA.UTIL. BEANCONTAINER<ID, BT>

- In-memory container for beans
  - All changes are lost
- Uses bean items
  - Generic bean type
  - Generic identifier type
    - `getItemIds() → List<ID>;`
    - `getItem(ID) → BeanItem<BT>`
  - Support for bean-to-id resolver
- Filterable, indexed, ordered and sortable
  - And broadcasts events
- No support for adding or removing properties

COM.VAADIN.DATA.UTIL. ABSTRACTINMEMORYCONTAINER <ITEM_ID, PROP_ID, ITEM_CLASS EXTENDS ITEM>

- Abstract in-memory container ☺
  - Indexed and ordered
  - Support for sorting and filtering
    - Some methods available
    - Not explicitly available
  - `Container.ItemSetChangeNotifier`
- Generic
  - Item identifier
  - Property identifier
  - Base item class

COM.VAADIN.DATA.UTIL. {FOO}WRAPPER

- Wrapper classes
  - Provide `Container.{foo}` when not available
  - `ContainerHierarchicalWrapper`
    - Adds hierarchy
  - `ContainerOrderedWrapper`
    - Adds order
  - `HierarchicalContainerOrderedWrapper`
    - Adds order to hierarchy
- Source of confusion
  - Certain UI components use wrappers internally
    - `setContainerDataSource != getContainerDataSource`
  - In your container always implement as much as possible ☺

COM.VAADIN.DATA.UTIL. INDEXEDCONTAINER

- Reference implementation
  - Indexed, ordered, sortable and filterable
  - Broadcasts all events
- In-memory container
- Any item id, any property id, any item
- One subclass
  - `HierarchicalContainer`
    - With hierarchy

COM.VAADIN.DATA.UTIL. SQLCONTAINER.<u>SQLCONTAINER</u>

- Any SQL database
- Two modes
  - Tables
    - Requires version column in the table
      - Must be updated automatically by the database
    - All features out of the box
      - <u>Except</u> auto-fetching database-generated values
  - Free queries
    - Statement delegate to support filtering, sorting, …
- Quite an impressive list of limitations

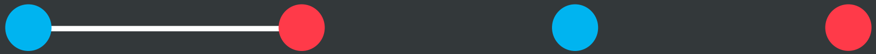
vaadin}>

JPACONTAINER (FREE ADD-ON)

- Container for JPA
  - Lazy loading
  - Filtering
  - Nested properties
  - Caching
- Works with Hibernate and EclipseLink
- Almost out-of-the-box CRUD
- JPA = Object-relational mapping
  - Tables = classes
  - Rows = objects
  - Columns = bean properties

FIELD BINDING

What to do with the data source



COM.VAADIN.DATA.FIELDGROUP.FIELDGROUP

- Binds fields with properties
  - Not a component
    - Does not manage layout
    - Cannot be added to other components
  - Handles property value changes of an item
    - Should be `Item.Viewer`
      - Methods are there, but not the interface
- Commits and discards changes
- Supports building fields
  - `FieldGroupFieldFactory`
    - `DefaultFieldGroupFieldFactory`
  - Fields need to be added to a layout

COM.VAADIN.UI.FIELD<T>

- Component for `Property<T>`
  - Property type depends on a component
    - Text fields → strings
    - Date fields → date
  - Most components are fields
    - Easily connected to a data source
- Is a property
  - Broadcasts events
- Can be required
  - With a custom error message
- Can be buffered and validated

COM.VAADIN.DATA.BUFFERED

- Buffered
  - All changes are buffered locally until committed
  - Read-through
    - Value read is up to date with the source
  - Write-through
    - Changes are immediately written to the source
- Commit
  - Writes changes since the last commit
- Discard
  - Restores state of the last commit
- All fields are buffered

COM.VAADIN.DATA. VALIDATABLE

- Maintains a list of validators
  - `com.vaadin.data.Validator`
    - `validate(Object) throws InvalidValueException;`
- Allows or disallows invalid values
- Defines two ways of validation
  - `validate() throws InvalidValueException;`
  - `isValid() → boolean;`
- No methods for setting or getting value
- One subinterface
  - `BufferedValidatable`
  - Allows or disallows comitting invalid values
  - Implemented by all fields
- Few useful implementations in the Framework

DEMO!

Shoutbox step 4
http://github.com/vaadin-miki/shoutbox

end branch: `step-04`

THE PLAN

- Add static container for messages
  - Shared across the VM
- Submit → add item to container
- Listen to events
  - Needs to be done in UI constructor
- Push the changes to clients
  - `@Push` on the UI
  - `this.access(new Runnable(... push();))`
- ???
- Profit!

SUMMARY

What did we do today

LESSONS OF TODAY (HOPEFULLY)

- Vaadin Data Model
  - What are the key elements?
  - How to link it to components?
- Events
  - How to catch events?
- Notifications and windows
  - How to display annoying popups?
- Server Push

COMING UP NEXT

- Styling, layouts, navigation
- Extending Vaadin

THE END

SUGGESTIONS?
QUESTIONS?

miki@vaadin.com
t: @mikiolsz