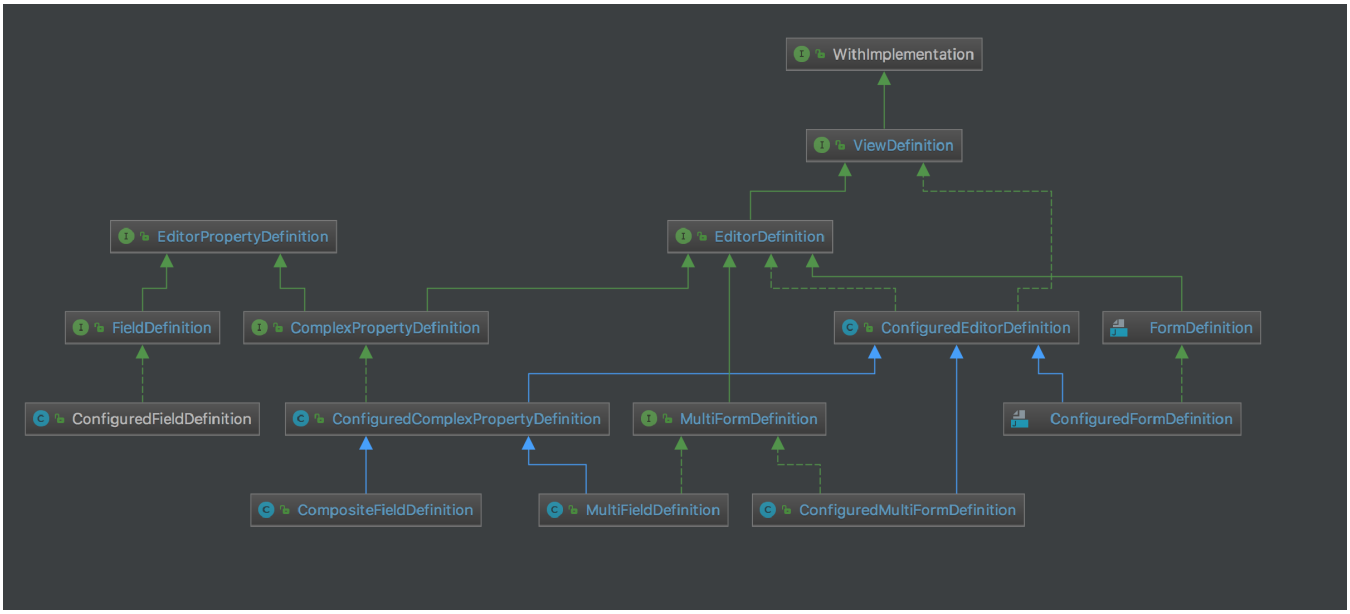
Databinding improvements

Main points

- Separation of the form model from the layout
- Keeping the transient changes directly in Vaadin fields, avoiding attempts to replicate JCR session across requests (aka JCRNodeAdapter)
- Implementation of the complex fields (composites, multi-fields etc) based on the form/editor concept
- Vaadin 8 powered implementation

Definitions



EditorDefinition

```
public interface EditorDefinition<T> extends ViewDefinition<EditorView<T>> {
}
```

EditorView

```
/**
 * Base interface of an editor.
 */
public interface EditorView<T> extends UiFrameworkView {

    void populate(T item);

    void write(T item);

    List<BinderValidationStatus<?>> validate();
}
```

FormDefinition

```
public interface FormDefinition<T> extends EditorDefinition<T> {

    List<EditorPropertyDefinition> getProperties();

    LayoutDefinition getLayout();

    ...
}
```

Comments on the structure:

- The only thing base `EditorDefinition` enforces is that the related view is an editor view to extend interface `EditorView`
- The latter does not enforce any particular structure of the implementation, just basic data-binding capabilities
- `FormDefinition` and the related `FormView` are an extension of the editor concept. Form assumes the presence of the properties and the layout definition.
- The `EditorPropertyDefinition` is either a 'simple' `FieldDefinition` or a `ComplexPropertyDefinition` which is a some form of editor view definition plus the definition of the strategy of obtaining the child item/node the complex field to bind to.

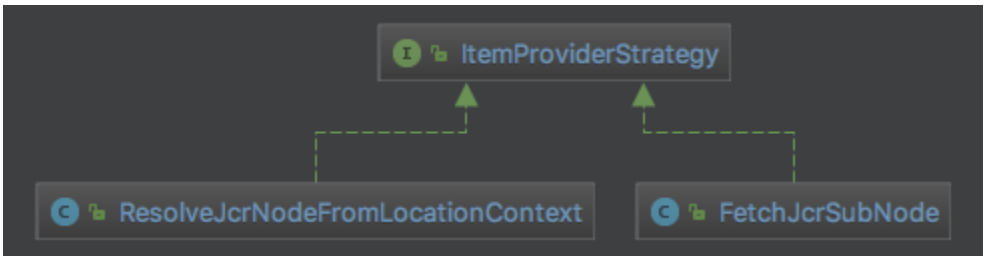
ItemProviderStrategy

ItemProviderStrategy

```
public interface ItemProviderStrategy<T> {

    Optional<T> read();
}
```

Currently there's only several impls of such strategy and all related to JCR, though this seems to be enough to do the trick for the data binding.



Note: the interface might still change!

Multiforms

I18NSupport

Cross-field validation/dependencies