# Factories:

Code:

public class ViewModelTextFieldTableCellVisualizationFactory<**S, T**> implements Callback<TableColumn<**S, T**>**,** TableCell<**S, T**>> {  
  
 private static final PseudoClass *INVALID\_PSEUDO\_CLASS* = PseudoClass.*getPseudoClass*("invalid")**;** private Function<**S,** ValidationStatus> validationStatusProperty**;** private StringConverter<**T**> stringConverter**;** public ViewModelTextFieldTableCellVisualizationFactory<**S, T**> withValidation(Function<**S,** ValidationStatus> validationStatusProperty) {  
 this.validationStatusProperty = validationStatusProperty**;** return this**;** }  
  
 public void install(TableColumn<**S, T**> column**,** StringConverter<**T**> stringConverter) {  
 column.setCellFactory(this)**;** this.stringConverter = stringConverter**;** }

Location:

src/main/java/org/jabref/gui/util/ViewModelTextFieldTableCellVisualizationFactory.java

Reasoning:

Paired with src/main/java/org/jabref/gui/util/ValueTableCellFactory.java

Code:

*/\*\*  
 \* Constructs a {****@link*** *TableCell} based on the value of the cell and a bunch of specified converter methods.  
 \*  
 \** ***@param*** <*S*> *view model of table row  
 \** ***@param*** <*T*> *cell value  
 \*/*public class ValueTableCellFactory<**S, T**> implements Callback<TableColumn<**S, T**>**,** TableCell<**S, T**>> {  
  
 private Function<**T,** String> toText**;** private BiFunction<**S, T,** Node> toGraphic**;** private BiFunction<**S, T,** EventHandler<? super MouseEvent>> toOnMouseClickedEvent**;** private Function<**T,** BooleanExpression> toDisableExpression**;** private Function<**T,** BooleanExpression> toVisibleExpression**;** private BiFunction<**S, T,** String> toTooltip**;** private Function<**T,** ContextMenu> contextMenuFactory**;** private BiFunction<**S, T,** ContextMenu> menuFactory**;** public ValueTableCellFactory<**S, T**> withText(Function<**T,** String> toText) {  
 this.toText = toText**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withGraphic(Function<**T,** Node> toGraphic) {  
 this.toGraphic = (rowItem**,** value) -> toGraphic.apply(value)**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withGraphic(BiFunction<**S, T,** Node> toGraphic) {  
 this.toGraphic = toGraphic**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withTooltip(BiFunction<**S, T,** String> toTooltip) {  
 this.toTooltip = toTooltip**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withTooltip(Function<**T,** String> toTooltip) {  
 this.toTooltip = (rowItem**,** value) -> toTooltip.apply(value)**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withOnMouseClickedEvent(BiFunction<**S, T,** EventHandler<? super MouseEvent>> toOnMouseClickedEvent) {  
 this.toOnMouseClickedEvent = toOnMouseClickedEvent**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withOnMouseClickedEvent(Function<**T,** EventHandler<? super MouseEvent>> toOnMouseClickedEvent) {  
 this.toOnMouseClickedEvent = (rowItem**,** value) -> toOnMouseClickedEvent.apply(value)**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withDisableExpression(Function<**T,** BooleanExpression> toDisableBinding) {  
 this.toDisableExpression = toDisableBinding**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withVisibleExpression(Function<**T,** BooleanExpression> toVisibleBinding) {  
 this.toVisibleExpression = toVisibleBinding**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withContextMenu(Function<**T,** ContextMenu> contextMenuFactory) {  
 this.contextMenuFactory = contextMenuFactory**;** return this**;** }  
  
 public ValueTableCellFactory<**S, T**> withMenu(BiFunction<**S, T,** ContextMenu> menuFactory) {  
 this.menuFactory = menuFactory**;** return this**;** }

Location:

src/main/java/org/jabref/gui/util/ValueTableCellFactory.java

Reasoning:

Paired with src/main/java/org/jabref/gui/util/ViewModelTextFieldTableCellVisualizationFactory.java

# Builder:

Code:

public class BibEntryTypeBuilder {  
  
 private EntryType type = StandardEntryType.*Misc***;** private Set<BibField> fields = new LinkedHashSet<>()**;** private Set<OrFields> requiredFields = new LinkedHashSet<>()**;** public BibEntryTypeBuilder withType(EntryType type) {  
 this.type = type**;** return this**;** }  
  
 public BibEntryTypeBuilder withImportantFields(Set<BibField> newFields) {  
 return withImportantFields(newFields.stream().map(BibField::getField).collect(Collectors.*toCollection*(LinkedHashSet::new)))**;** }  
  
 public BibEntryTypeBuilder withImportantFields(Collection<Field> newFields) {  
 this.fields = Streams.*concat*(fields.stream()**,** newFields.stream().map(field -> new BibField(field**,** FieldPriority.*IMPORTANT*)))  
 .collect(Collectors.*toCollection*(LinkedHashSet::new))**;** return this**;** }  
  
 public BibEntryTypeBuilder withImportantFields(Field... newFields) {  
 return withImportantFields(Arrays.*asList*(newFields))**;** }  
  
 public BibEntryTypeBuilder withDetailFields(Collection<Field> newFields) {  
 this.fields = Streams.*concat*(fields.stream()**,** newFields.stream().map(field -> new BibField(field**,** FieldPriority.*DETAIL*)))  
 .collect(Collectors.*toCollection*(LinkedHashSet::new))**;** return this**;** }  
  
 public BibEntryTypeBuilder withDetailFields(Field... fields) {  
 return withDetailFields(Arrays.*asList*(fields))**;** }  
  
 public BibEntryTypeBuilder withRequiredFields(Set<OrFields> requiredFields) {  
 this.requiredFields = requiredFields**;** return this**;** }  
  
 public BibEntryTypeBuilder withRequiredFields(Field... requiredFields) {  
 this.requiredFields = Arrays.*stream*(requiredFields).map(OrFields::new).collect(Collectors.*toCollection*(LinkedHashSet::new))**;** return this**;** }  
  
 public BibEntryTypeBuilder withRequiredFields(OrFields first**,** Field... requiredFields) {  
 this.requiredFields = Stream.*concat*(Stream.*of*(first)**,** Arrays.*stream*(requiredFields).map(OrFields::new)).collect(Collectors.*toCollection*(LinkedHashSet::new))**;** return this**;** }  
  
 public BibEntryTypeBuilder withRequiredFields(List<OrFields> first**,** Field... requiredFields) {  
 this.requiredFields = Stream.*concat*(first.stream()**,** Arrays.*stream*(requiredFields).map(OrFields::new)).collect(Collectors.*toCollection*(LinkedHashSet::new))**;** return this**;** }  
  
 public BibEntryType build() {  
 // Treat required fields as important ones  
 Stream<BibField> requiredAsImportant = requiredFields.stream()  
 .flatMap(Set::stream)  
 .map(field -> new BibField(field**,** FieldPriority.*IMPORTANT*))**;** Set<BibField> allFields = Stream.*concat*(fields.stream()**,** requiredAsImportant).collect(Collectors.*toCollection*(LinkedHashSet::new))**;** return new BibEntryType(type**,** allFields**,** requiredFields)**;** }  
}

Location:

src/main/java/org/jabref/model/entry/BibEntryTypeBuilder.java

Reasoning:

Multiple construction methods under same method