**Singleton:**

Code:

// This String is used in the encoded list in prefs of external file type  
// modifications, in order to indicate a removed default file type:  
private static final String *FILE\_TYPE\_REMOVED\_FLAG* = "REMOVED"**;**// The only instance of this class:  
private static ExternalFileTypes *singleton***;**

Location:

src/main/java/org/jabref/gui/externalfiletype/ExternalFileTypes.java

Reasoning:

//A singleton design pattern occurs when a class is only instanced once.

**Factories:**

Code:

*/\*\*  
 \* Constructs a {****@link*** *TableCell} based on an optional value of the cell and a bunch of specified converter methods.  
 \*  
 \** ***@param*** <*S*> *view model of table row  
 \** ***@param*** <*T*> *cell value  
 \*/*public class OptionalValueTableCellFactory<**S, T**> extends ValueTableCellFactory<**S,** Optional<**T**>> {  
  
 private BiFunction<**S, T,** Node> toGraphicIfPresent**;** private Node defaultGraphic**;** public OptionalValueTableCellFactory<**S, T**> withGraphicIfPresent(BiFunction<**S, T,** Node> toGraphicIfPresent) {  
 this.toGraphicIfPresent = toGraphicIfPresent**;** setToGraphic()**;** return this**;** }  
  
 public OptionalValueTableCellFactory<**S, T**> withDefaultGraphic(Node defaultGraphic) {  
 this.defaultGraphic = defaultGraphic**;** setToGraphic()**;** return this**;** }  
  
 private void setToGraphic() {  
 withGraphic((rowItem**,** item) -> {  
 if (item.isPresent() && toGraphicIfPresent != null) {  
 return toGraphicIfPresent.apply(rowItem**,** item.get())**;** } else {  
 return defaultGraphic**;** }  
 })**;** }  
}

Location:

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

Reasoning:

Due to having optional fields, this implementation of the abstract method actually implements different specifications of the same object. It should instead all be agreggated in a single method instead of two.

**Builder:**

Code:

public static class Builder {  
  
 private Path initialDirectory**;** public DirectoryDialogConfiguration build() {  
 return new DirectoryDialogConfiguration(initialDirectory)**;** }  
  
 public Builder withInitialDirectory(Path directory) {  
  
 directory = directory.toAbsolutePath()**;** // Dir must be a folder, not a file  
 if (!Files.*isDirectory*(directory)) {  
 directory = directory.getParent()**;** }  
 // The lines above work also if the dir does not exist at all!  
 // NULL is accepted by the filechooser as no inital path  
  
 if (!Files.*exists*(directory)) {  
  
 directory = null**;** }  
 initialDirectory = directory**;** return this**;** }  
  
 public Builder withInitialDirectory(String directory) {  
 withInitialDirectory(Path.*of*(directory))**;** return this**;** }  
 }  
}

Location:

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

Reasoning:

The intent of the Builder design pattern is to separate the construction of a complex object from its representation.