EditIntegration refactor plan

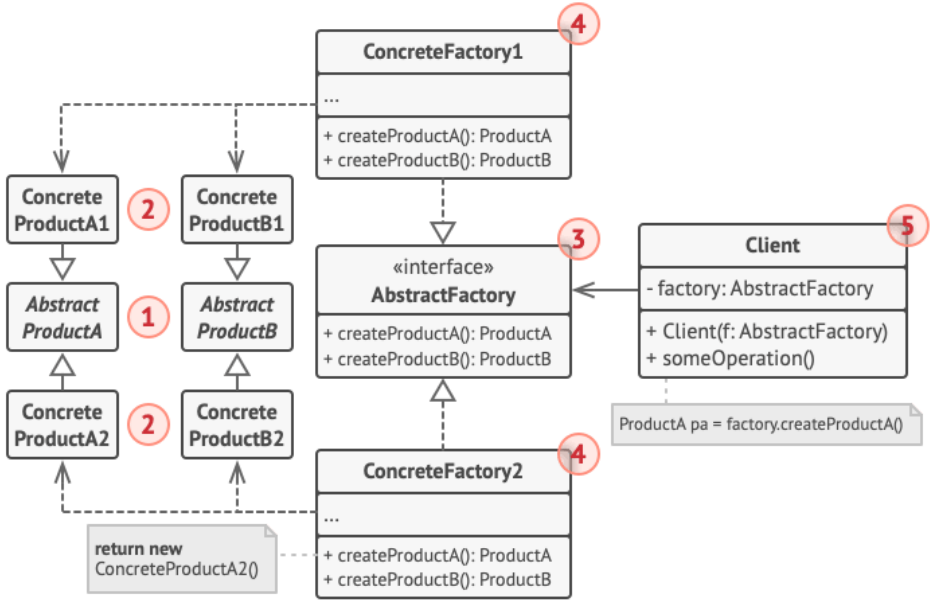
Introduction

Utilizing “Abstract Factory”

- EditIntegration form has multiple variants
  - Integration
  - Interface
  - Connection
- Each variant has multiple sub-variants for each service type
  - webMethods
  - ActiveTransfer
  - SAP PO
  - webMethods IO
  - Etc.

What is Abstract Factory

- Creational Design pattern
- Create “Group of Related object” without specifying concrete class
  - Related object => Integration stage form for each platform type



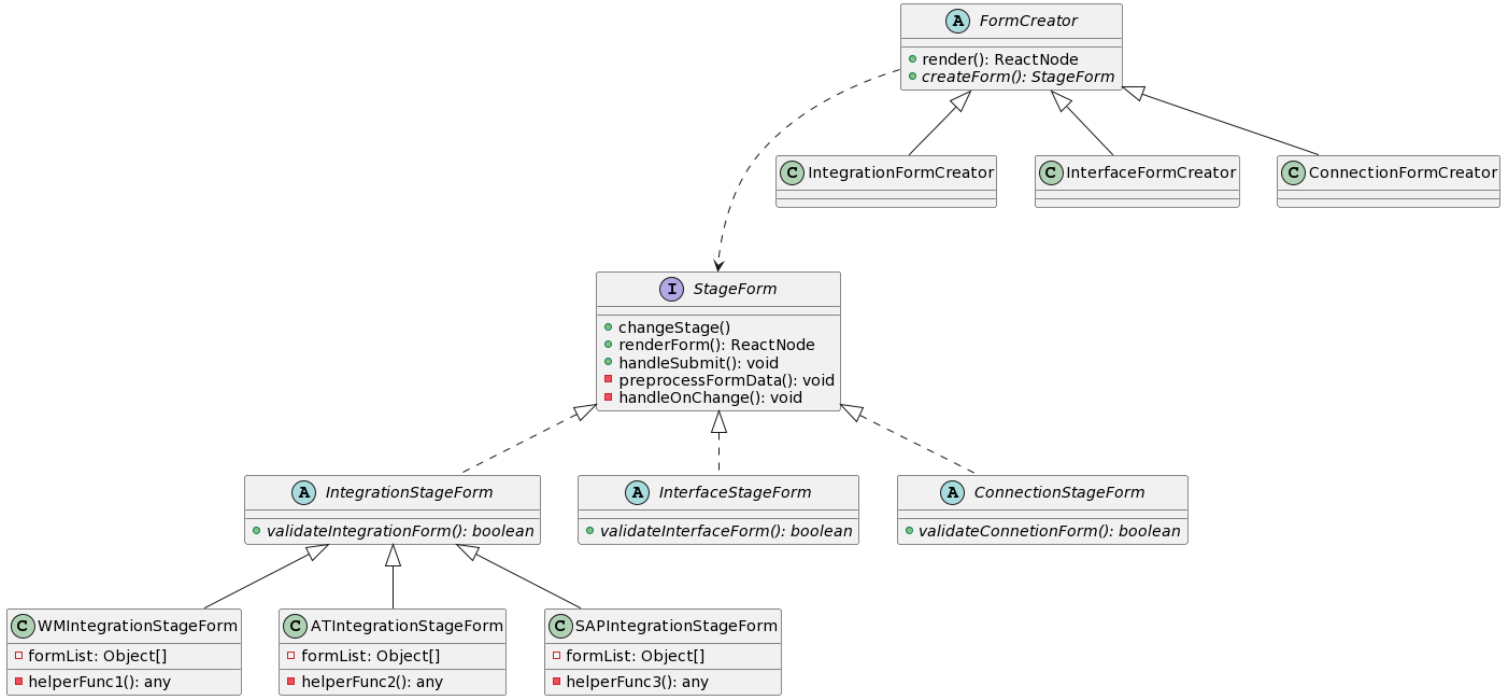
Source: https://refactoring.guru/design-patterns/abstract-factory

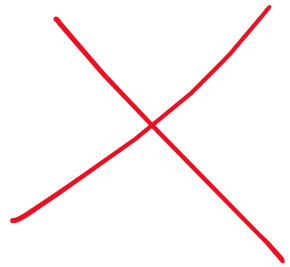
Design progress

PlantUML script are provided.

Factory (Friday idea) => Don’t suit the use case

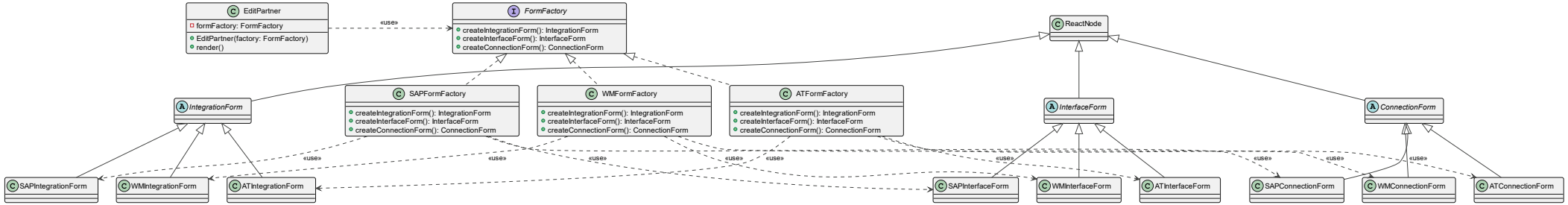
```
@startuml
interface StageForm{
   + changeStage()
   + renderForm(): ReactNode
   + handleSubmit(): void
   - preprocessFormData(): void
   - handleOnChange(): void
}

abstract class IntegrationStageForm implements StageForm{
   + {abstract} validateIntegrationForm(): boolean
}
abstract class InterfaceStageForm implements StageForm{
   + {abstract} validateInterfaceForm(): boolean
}
abstract class ConnectionStageForm implements StageForm{
    + {abstract} validateConnetionForm(): boolean
}

class WMIntegrationStageForm extends IntegrationStageForm {
   - formList: Object[]
   - helperFunc1(): any
}
class ATIntegrationStageForm extends IntegrationStageForm {
   - formList: Object[]
   - helperFunc2(): any
}
class SAPIntegrationStageForm extends IntegrationStageForm {
   - formList: Object[]
   - helperFunc3(): any
}


abstract class FormCreator{
   + render(): ReactNode
   + {abstract} createForm(): StageForm
}
class IntegrationFormCreator extends FormCreator{}
class InterfaceFormCreator extends FormCreator{}
class ConnectionFormCreator extends FormCreator{}

FormCreator ...> StageForm
@enduml
```





Abstract Factory (Monday idea)

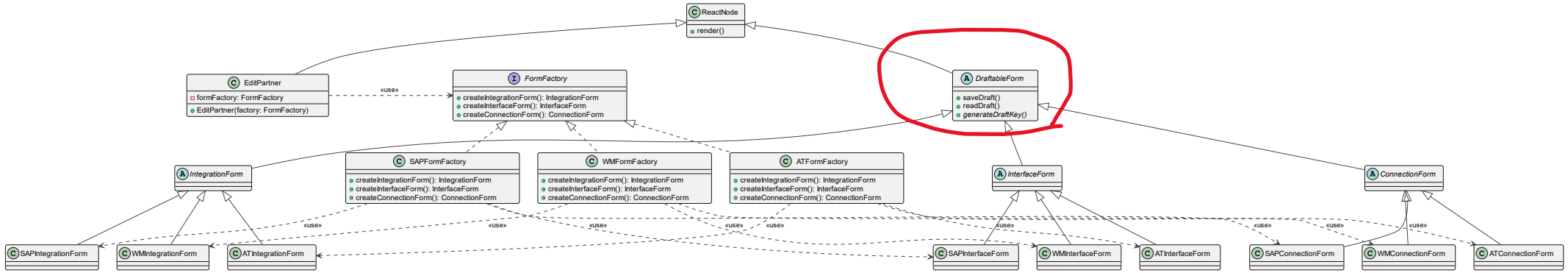
```
@startuml
!theme vibrant
class ReactNode{}
class EditPartner{
  - formFactory: FormFactory

  + EditPartner(factory: FormFactory)
  + render()
}

together {
  interface FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class WMFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class ATFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class SAPFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
}


together {
  abstract class IntegrationForm extends ReactNode{}
  class WMIntegrationForm extends IntegrationForm{}
  class ATIntegrationForm extends IntegrationForm{}
  class SAPIntegrationForm extends IntegrationForm{}
}
together {
  abstract class InterfaceForm extends ReactNode{}
  class WMInterfaceForm extends InterfaceForm{}
  class ATInterfaceForm extends InterfaceForm{}
  class SAPInterfaceForm extends InterfaceForm{}
}
together {
  abstract class ConnectionForm extends ReactNode{}
  class WMConnectionForm extends ConnectionForm{}
  class ATConnectionForm extends ConnectionForm{}
  class SAPConnectionForm extends ConnectionForm{}
}

EditPartner .Right.> FormFactory: <<use>>

WMFormFactory .down.> WMIntegrationForm: <<use>>
WMFormFactory .down.> WMInterfaceForm: <<use>>
WMFormFactory .down.> WMConnectionForm: <<use>>

ATFormFactory .down.> ATIntegrationForm: <<use>>
ATFormFactory .down.> ATInterfaceForm: <<use>>
ATFormFactory .down.> ATConnectionForm: <<use>>

SAPFormFactory .down.> SAPIntegrationForm: <<use>>
SAPFormFactory .down.> SAPInterfaceForm: <<use>>
SAPFormFactory .down.> SAPConnectionForm: <<use>>
@enduml
```



```
@startuml
!theme vibrant
class ReactNode{
  + render()
}
class EditPartner extends ReactNode{
  - formFactory: FormFactory

  + EditPartner(factory: FormFactory)
}

together {
  interface FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class WMFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class ATFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class SAPFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
}

abstract class DraftableForm extends ReactNode {
  + saveDraft()
  + readDraft()
  + {abstract} generateDraftKey()
}

together {
  abstract class IntegrationForm extends DraftableForm {}
  class WMIntegrationForm extends IntegrationForm{}
  class ATIntegrationForm extends IntegrationForm{}
  class SAPIntegrationForm extends IntegrationForm{}
}
together {
  abstract class InterfaceForm extends DraftableForm {}
  class WMInterfaceForm extends InterfaceForm{}
  class ATInterfaceForm extends InterfaceForm{}
  class SAPInterfaceForm extends InterfaceForm{}
}
together {
  abstract class ConnectionForm extends DraftableForm {}
  class WMConnectionForm extends ConnectionForm{}
  class ATConnectionForm extends ConnectionForm{}
  class SAPConnectionForm extends ConnectionForm{}
}

EditPartner .Right.> FormFactory: <<use>>

WMFormFactory .down.> WMIntegrationForm: <<use>>
WMFormFactory .down.> WMInterfaceForm: <<use>>
WMFormFactory .down.> WMConnectionForm: <<use>>

ATFormFactory .down.> ATIntegrationForm: <<use>>
ATFormFactory .down.> ATInterfaceForm: <<use>>
ATFormFactory .down.> ATConnectionForm: <<use>>

SAPFormFactory .down.> SAPIntegrationForm: <<use>>
SAPFormFactory .down.> SAPInterfaceForm: <<use>>
SAPFormFactory .down.> SAPConnectionForm: <<use>>
@enduml
```

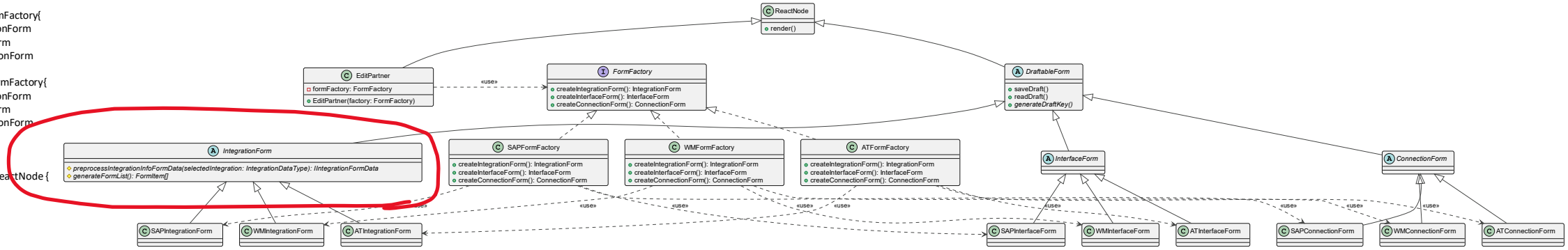
Friday Morning Class Diagram

- Added DraftableForm abstract class
  - saveDraft(),readDraft() is implemented here



Friday Afternoon Class Diagram

- Added abstract method to IntegrationForm
  - generateFormList()
  - preprocessIntegrationInfoFormData

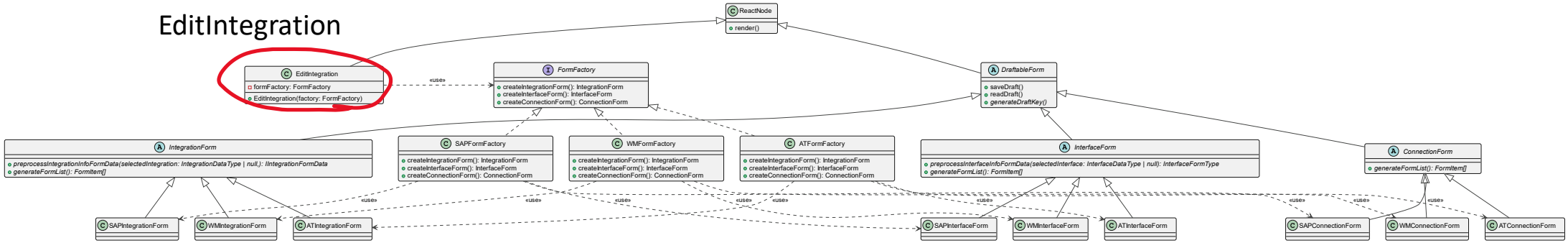


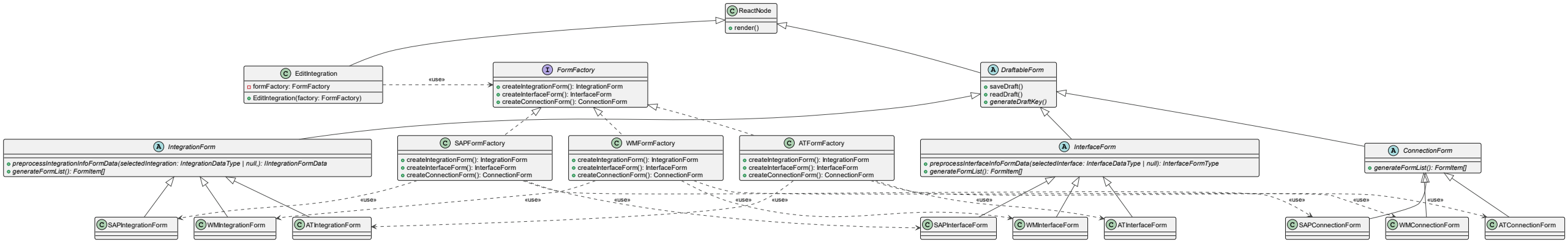

```
@startuml
!theme vibrant
class ReactNode{
  + render()
}
class EditPartner extends ReactNode{
  - formFactory: FormFactory

  + EditPartner(factory: FormFactory)
}

together {
  interface FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class WMFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class ATFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class SAPFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
}

abstract class DraftableForm extends ReactNode {
  + saveDraft()
  + readDraft()
  + {abstract} generateDraftKey()
}

together {
  abstract class IntegrationForm extends DraftableForm {
    # {abstract} preprocessIntegrationInfoFormData(selectedIntegration: IntegrationDataType): IIntegrationFormData
    # {abstract} generateFormList(): FormItem[]
  }
  class WMIntegrationForm extends IntegrationForm{}
  class ATIntegrationForm extends IntegrationForm{}
  class SAPIntegrationForm extends IntegrationForm{}
}
together {
  abstract class InterfaceForm extends DraftableForm {}
  class WMInterfaceForm extends InterfaceForm{}
  class ATInterfaceForm extends InterfaceForm{}
  class SAPInterfaceForm extends InterfaceForm{}
}
together {
  abstract class ConnectionForm extends DraftableForm {}
  class WMConnectionForm extends ConnectionForm{}
  class ATConnectionForm extends ConnectionForm{}
  class SAPConnectionForm extends ConnectionForm{}
}

EditPartner .Right.> FormFactory: <<use>>

WMFormFactory .down.> WMIntegrationForm: <<use>>
WMFormFactory .down.> WMInterfaceForm: <<use>>
WMFormFactory .down.> WMConnectionForm: <<use>>

ATFormFactory .down.> ATIntegrationForm: <<use>>
ATFormFactory .down.> ATInterfaceForm: <<use>>
ATFormFactory .down.> ATConnectionForm: <<use>>

SAPFormFactory .down.> SAPIntegrationForm: <<use>>
```

```
@startuml
!theme vibrant
class ReactNode{
  + render()
}
class EditIntegration extends ReactNode{
  - formFactory: FormFactory

  + EditIntegration(factory: FormFactory)
}

together {
  interface FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class WMFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class ATFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
  class SAPFormFactory implements FormFactory{
    + createIntegrationForm(): IntegrationForm
    + createInterfaceForm(): InterfaceForm
    + createConnectionForm(): ConnectionForm
  }
}

abstract class DraftableForm extends ReactNode {
  + saveDraft()
  + readDraft()
  + {abstract} generateDraftKey()
}

together {
  abstract class IntegrationForm extends DraftableForm {
    + {abstract} preprocessIntegrationInfoFormData(selectedIntegration: IntegrationDataType | null,): IIntegrationFormData
                 + {abstract} generateFormList(): FormItem[]
  }
  class WMIntegrationForm extends IntegrationForm{}
  class ATIntegrationForm extends IntegrationForm{}
  class SAPIntegrationForm extends IntegrationForm{}
}
together {
  abstract class InterfaceForm extends DraftableForm {
                 + {abstract} preprocessInterfaceInfoFormData(selectedInterface: InterfaceDataType | null): InterfaceFormType
                 + {abstract} generateFormList(): FormItem[]
  }
  class WMInterfaceForm extends InterfaceForm{}
  class ATInterfaceForm extends InterfaceForm{}
  class SAPInterfaceForm extends InterfaceForm{}
}
together {
  abstract class ConnectionForm extends DraftableForm {
                 + {abstract} generateFormList(): FormItem[]
  }
  class WMConnectionForm extends ConnectionForm{}
  class ATConnectionForm extends ConnectionForm{}
  class SAPConnectionForm extends ConnectionForm{}
}

EditIntegration .Right.> FormFactory: <<use>>

WMFormFactory .down.> WMIntegrationForm: <<use>>
WMFormFactory .down.> WMInterfaceForm: <<use>>
WMFormFactory .down.> WMConnectionForm: <<use>>
```

Final Diagram
- ConnectionForm will be implemented differently

Renamed to EditIntegration




ReactNode
render()
EditIntegration
formFactory: FormFactory
EditIntegration(factory: FormFactory)
«use»
FormFactory
createIntegrationForm(): IntegrationForm
createInterfaceForm(): InterfaceForm
createConnectionForm(): ConnectionForm
DraftableForm
saveDraft()
readDraft()
generateDraftKey()
IntegrationForm
preprocessIntegrationInfoFormData(selectedIntegration: IntegrationDataType | null,): IIntegrationFormData
generateFormList(): FormItem[]
SAPFormFactory
createIntegrationForm(): IntegrationForm
createInterfaceForm(): InterfaceForm
createConnectionForm(): ConnectionForm
WMFormFactory
createIntegrationForm(): IntegrationForm
createInterfaceForm(): InterfaceForm
createConnectionForm(): ConnectionForm
ATFormFactory
createIntegrationForm(): IntegrationForm
createInterfaceForm(): InterfaceForm
createConnectionForm(): ConnectionForm
InterfaceForm
preprocessInterfaceInfoFormData(selectedInterface: InterfaceDataType | null): InterfaceFormType
generateFormList(): FormItem[]
ConnectionForm
generateFormList(): FormItem[]
SAPIntegrationForm
WMIntegrationForm
ATIntegrationForm
SAPInterfaceForm
WMInterfaceForm
ATInterfaceForm
SAPConnectionForm
WMConnectionForm
ATConnectionForm

Summary

- XXFormFactory abstract class implemented FormFactory interface
- XXForm class extends DraftableForm abstract class
- DraftableForm is generic class with 2 params <P extends DraftableFormProps, S extends DraftableFormState>
  - Implemented this way to enable ability to add more prop/state to sub-class
- Usually, XXForm will render ReusableForm (more about this in the next section)
- XXForm will implemented all abstract method

Notes

Regarding useEffect, useState, useContext

- Class component can't use React Hook
- For useEffect hook, use component lifecycle method to update the data
- For useState hook, use this.state and setState() to update state value
- For useContext hook, specified contextType and use with this.context

Regarding context

- Class component can only use one context
- To use multiple context in class component,
  - create a provider that combined multiple contexts together.
  - Wrapped the component that will make use of combined context with the provider
  - Set contextType to be the created provider

```
const WMConnectionForm = () => (
    <SelectedIntegrationAndSelectedInterfaceProvider>
        <WMConnectionFormInner />
    </SelectedIntegrationAndSelectedInterfaceProvider>
)
You, 22 hours ago | 1 author (You)
class WMConnectionFormInner extends ConnectionForm {···
}
WMConnectionFormInner.contextType = SelectedIntegrationAndSelectedInterfaceContext
```

Other hooks in class component

- Design pattern is based on Object-Oriented concept
  - Must use class component to achieved
- Old components are functional component which utilized heavy usage of React Hook
  - Class component can’t use hook
- This link could be useful to make use of the hook in class component.
- https://www.glennstovall.com/how-to-use-useeffect-and-other-hooks-in-class-components/

Draft key generation

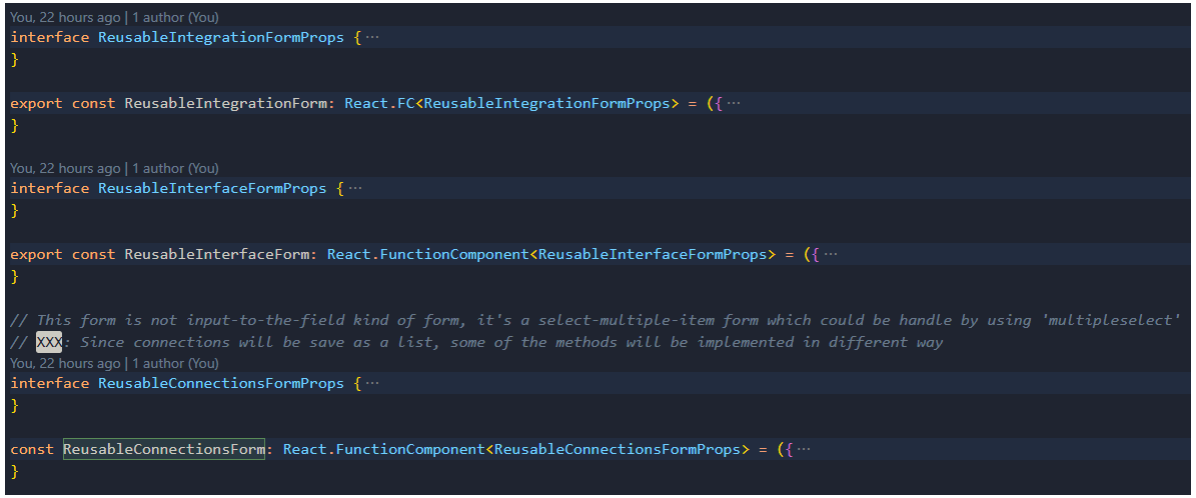
- Draft key generated from generateDraftKey() will be consumed by saveDraft() and readDraft()
- Suggested key pattern is the following
  - Prefix with partnerId
  - Each stage have more ID according to how it's nested
    - Integration stage -> integrationId
    - Interface stage -> integrationId, interfaceId
    - Start with the the most general item's ID and follow with more specific item's ID
      - GOOD: integrationId -> interfaceId
      - BAD: interfaceId -> integrationId
  - Suffix with <Technology><Stage>Draft
    - WMInterfaceDraft
    - ATIntegrationDraft
  - Join each part with "-"
  - Ex: (at webMethod interface stage) 1234-43-25-WMInterfaceDraft
    - 1234 = partnerId
    - 43 = integrationId
    - 25 = interfaceId
- Normally, the data to use with draft (from PoC), comes from context

Interacting with “unrelated draft”

- “Unrelated draft” = information that don’t directly belong to the form stage
  - Ex: selectedIntegration in Interface form stage
- The “unrelated draft” saveDraft(), readDraft(), generateDraftKey() function must be implemented on-demand and the implementation pattern (for these functions) is not enforced
  - But try to keep the pattern for easier maintenance work

ReusableForm

- For ease of development and higher reusablity, create ReusableForm component for each stage of editing
  - ReusableIntegrationForm
  - ReusableInterfaceForm
  - ReusableConnectionForm
- ReusableForm will be use for
  - Form rendering
  - Draft saving
  - Update integration and related data



- Should be return from render() of IntegrationForm, InterfaceForm, ConnectionForm

ReusableForm (cont'd)

- Since ReusableForm use useForm hook, it must be functional component.
- Hence, ReusableForm method implementation can't be enforced
- Check implementation example below to understand the intended behavior.
- Implementation: https://github.com/TanapolWong-asa/abstract-factory-form/blob/master/src/pages/EditPartner/Form/reusableForm.tsx

Additional Form for each stage

- In case of any stage required additional form,
  - Render it in class component along side with ReusableForm
  - Keep in mind the following functionalities
    - Form rendering
    - Draft saving
    - Update related data***

```
render() {
    return (
        <>
            <ReusableConnectionsForm
                saveDraft={this.saveDraft}
                readDraft={this.readDraft}
                formList={this.generateFormList()}
            />
            <SomeOtherForm />
        </>
    )
}
```

Breaking the pattern

- Some component might need a specific implementation
- First, try to follow the pattern.
- If couldn’t be done, ignore the pattern then…
  - Document what break the pattern
  - Document why you need to break the pattern

```
// This form is not input-to-the-field kind of form, it's a select-multiple-item form which could be handle by using 'multipleselect'
// XXX: Since connections will be save as a list, some of the methods will be implemented in different way
You, 22 hours ago | 1 author (You)
interface ReusableConnectionsFormProps {···
}

const ReusableConnectionsForm: React.FunctionComponent<ReusableConnectionsFormProps> = ({···
}
```