CS F213 - Object Oriented Programming

J. Jennifer Ranjani

email: jennifer.ranjani@pilani.bits-pilani.ac.in

Chamber: 6121 P, NAB

Consultation: Appointment by e-mail

https://github.com/JenniferRanjani/Object-Oriented-Programming-with-Java

BITS Pilani

Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Design Patterns

Introduction

- It is a general reusable solution or template to a commonly occurring problem in software design.
- Patterns show relationships and interactions between classes or objects.
- Goal:
  - Understand the problem and match it with some pattern
  - Reuse of old interface or making the present design reusable for the future usage

Types of Design Patterns

- Creational:
  - It deals about class instantiation or object creation.
  - Further classified into class creational patterns and object creational patterns.
  - Eg. Factory method, abstract factory, singleton and builder
- Structural:
  - It is about organizing different classes and objects to form larger structures and provide new functionality.
  - Eg. Adapter, Composite, Decorator, Proxy
- Behavioral:
  - It is about identifying common communication patterns between objects
  - Eg. Iterator, State, Strategy, Observer, Command, Chain of Responsibility

Creational Design Patterns

Singleton Pattern

- Singleton class is a class that has a single object. We need to ensure that no additional instances can be created accidentally.
  - Eg. Class that generate random numbers.

- Singleton class
  - Declare a static variable to keep the class's sole instance
  - Use private access modifier for class constructor, thus other classes can not create instance because they have no access to the constructor.
  - The class constructs a single instance of itself. Supply a static method that returns a reference to the single instance.
  - Use Synchronized keyword to protect more than one thread accessing the instance variable.
  - Override clone method and throw a CloneNotSupportedException so that another instance cannot be created by cloning the singleton object.

Factory Pattern

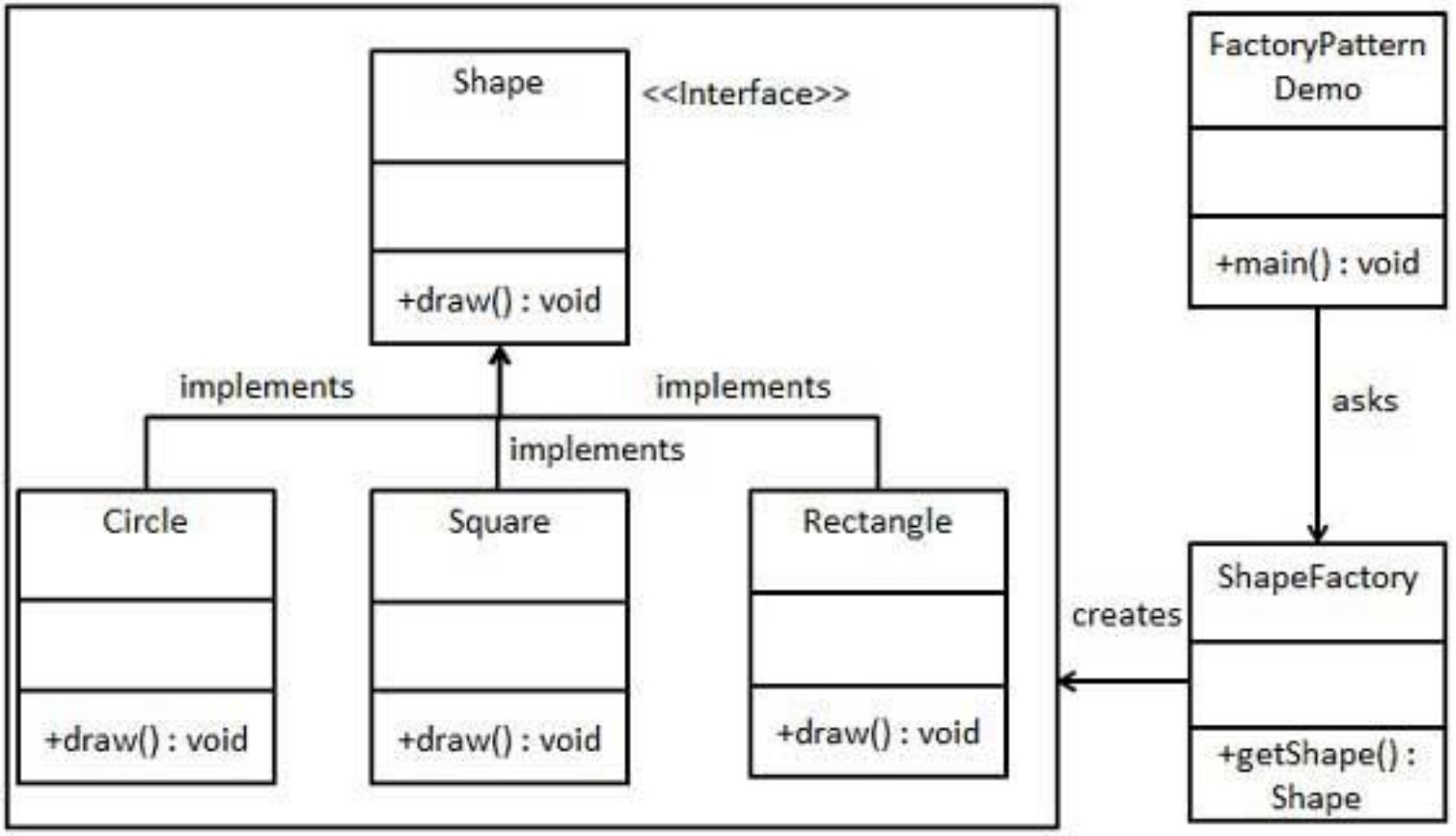
innovate achieve lead

- It is used when there is a super class with multiple sub-classes and dependent on the user input one of the sub classes object should be return.

- Creates objects without exposing the creation logic to the client and refers to the newly created object using a common interface

- Returns an instance of one of the several possible classes depending on the data provided to it

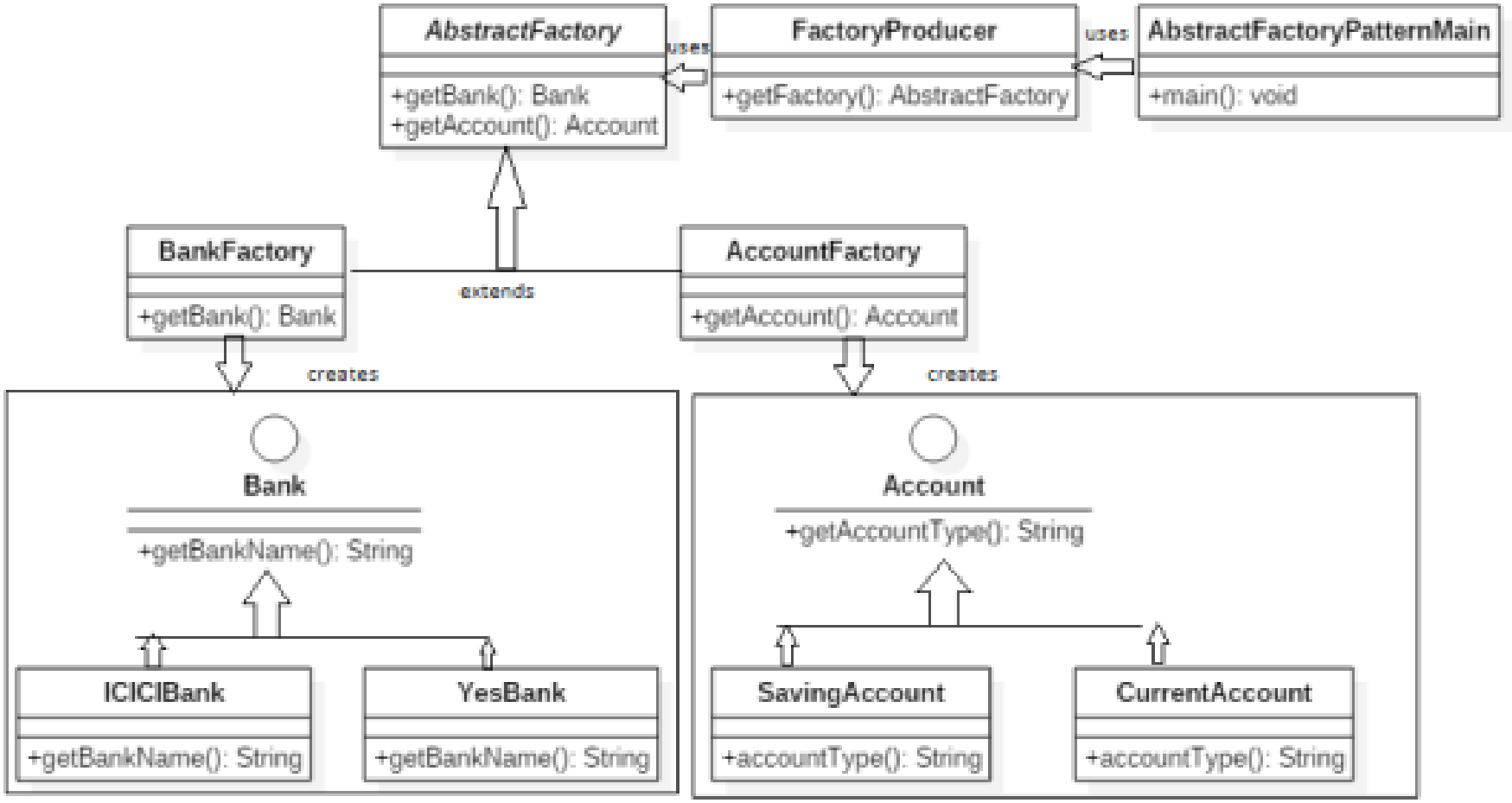
Factory Pattern – Example



Abstract Factory Pattern

- It is an interface and it is responsible for creating a factory of objects without explicitly specifying their classes.
- Each generated factory can give the objects as per the Factory pattern.
- It is called factory of factories.

Abstract Factory Pattern



Builder Pattern

- Factory or Abstract factory design patterns will not be suitable when the Object contains a lot of attributes.

- Issues:
  - Passing too many parameters from the client program to the factory class can be error prone because most of the time, the type of the parameters are same and it is difficult to maintain the order of the arguments at the client side.
  - Some parameters might be optional but we are forced to send NULL when factory pattern is used.

- Builder pattern solves the issues by providing a step by step way to build the object.

Pros and Cons

- Pros:
  - Code is more maintainable if the number of fields required to create an object is more than 4 or 5.
  - Object creational code is less error prone as the user will know what they are passing because of explicit method call
  - It increases robustness as only fully constructed object will be available to the client.

- Cons:
  - It is verbose and requires code duplication as the builder need to copy all fields from the original or item class.

Right time to use builder pattern

- Cake Analogy
  - Mandatory incredients – egg, milk, flour
  - Optional – cherry, fruits
  - Overloaded constructors is necessary when different kinds of cake is to be made; and they might accept many parameters.
- Problems:
  - Too many constructors to maintain
  - Error prone because many fields has same type
    - Both sugar and butter are measured in cups
    - If we pass, butter twice instead of sugar, the compiler wont complain

Steps for Builder design patter

- Make a static nested class for the builder.
- Builder has same set of fields as the original class.
- Builder.build() copies all the builder field into actual class and return the object of the item class
- Item class should have a private constructor to create its object from build and prevent outside access.