Object Oriented Analysis and Design with Java

UE19CS353

Dr. Geetha D
Department of Computer Science and Engineering

Acknowledgements: Significant portions of the information in the slide sets presented through the course in the class, are extracted from the prescribed text books, information from the Internet. Since these are only intended for presentation for teaching within PESU, there was no explicit permission solicited. We would like to sincerely thank and acknowledge that the credit/rights remain with the original authors/creators only

UE19CS353: Object Oriented Analysis and Design with Java

Design Patterns

Department of Computer Science and Engineering

What are Design Patterns?

- Design patterns are used to represent some of the best practices adapted by experienced object-oriented software developers
- Reusable solutions to the problems
- Interaction between the objects
- Template, not a solution
- Language independent

Design Patterns

- In software engineering, a design pattern is general reusable solution to commonly occurring problem in software design.
- It is not a finished design that can be transferred directly into code.
- Object-oriented design patterns typically show relationships and interactions between classes or objects, without specifying final application classes or objects that are involved.

Selection or Usage of Design Pattern

In 1994, four authors Erich Gamma, Richard Helm, Ralph Johnson und John Vlissides published a book titled Design Patterns - Elements of Reusable Object-Oriented Software which initiated the concept of Design Pattern in Software development.

These authors are collectively known as **Gang of Four (GOF).** According to these authors design patterns are primarily based on the following principles of object orientated design.

- Program to an interface not an implementation
- Favor object composition over inheritance

Selection or Usage of Design Pattern

Design Patterns have two main usages in software development.

- **Common platform for developers**

Design patterns provide a standard terminology and are specific to particular scenario. For example, a singleton design pattern signifies use of single object so all developers familiar with single design pattern will make use of single object and they can tell each other that program is following a singleton pattern.

- **Best Practices**

Design patterns have been evolved over a long period of time and they provide best solutions to certain problems faced during software development. Learning these patterns helps un-experienced developers to learn software design in an easy and faster way.

Why use Design Patterns?

Design Objectives

- Good Design (the "ilities")

- High readability and maintainability
- High extensibility
- High scalability
- High testability
- High reusability

Elements of a Design Pattern

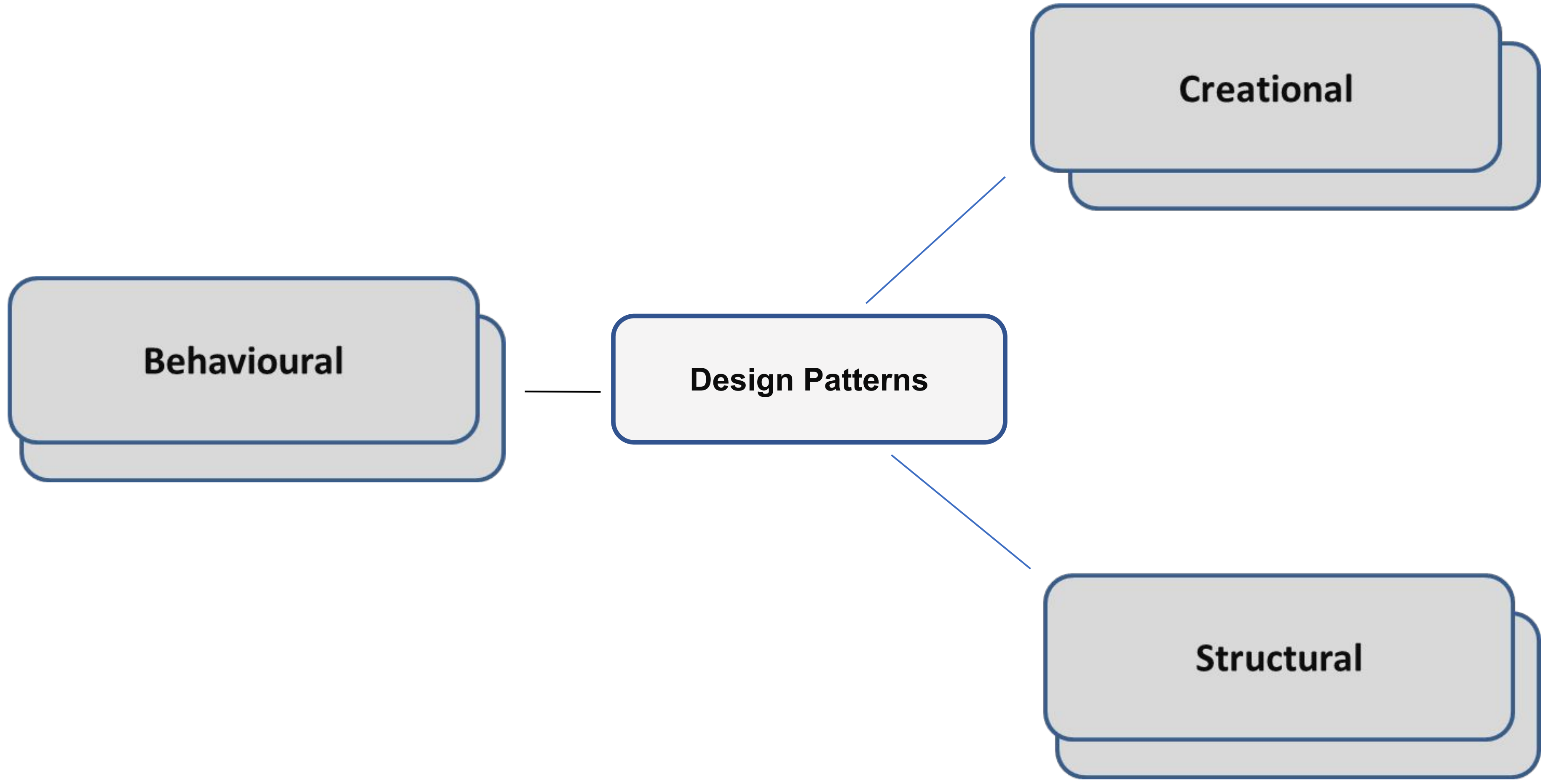
A pattern has four essential elements (GoF)

- **Name**
    - Describes the pattern
    - Adds to common terminology for facilitating communication (i.e. not just sentence enhancers)
- **Problem**
    - Describes when to apply the pattern
    - Answers - What is the pattern trying to solve?

Elements of a Design Pattern (cont.)

- **Solution**
    - Describes elements, relationships, responsibilities, and collaborations which make up the design
- **Consequences**
    - Results of applying the pattern
    - Benefits and Costs
    - Subjective depending on concrete scenarios

Design Patterns Classification



Design Patterns Classification

A design pattern can be classified as

Creational: These design patterns provide a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator. This gives program more flexibility in deciding which objects need to be created for a given use case.

Structural : These design patterns concern class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities

Behavioral: These design patterns are specifically concerned with communication between objects.

Pros/Cons of Design Patterns

Pros

- Add consistency to designs by solving similar problems the same way, independent of language
- Add clarity to design and design communication by enabling a common vocabulary
- Improve time to solution by providing templates which serve as foundations for good design
- Improve reuse through composition

Pros/Cons of Design Patterns

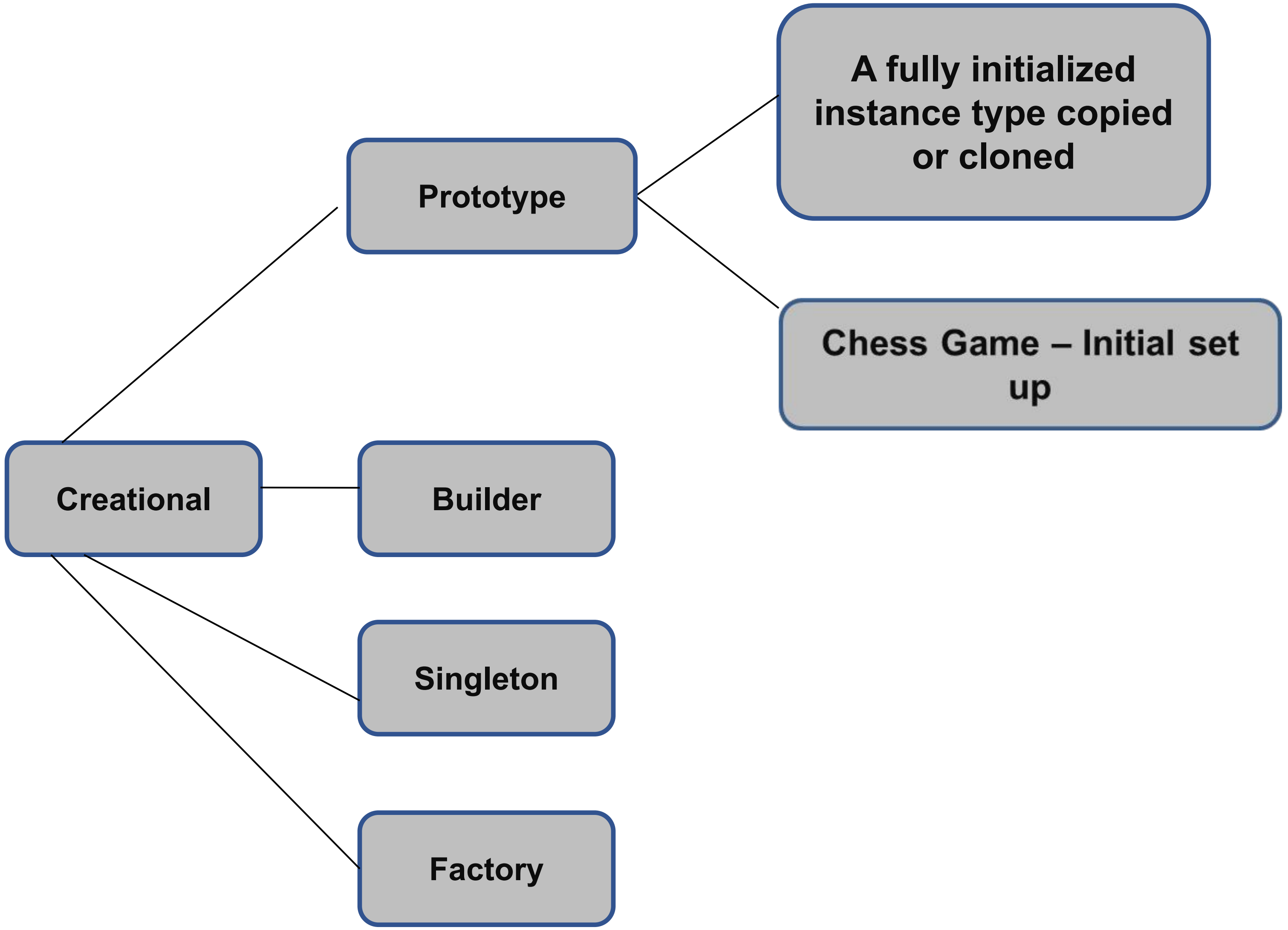
Cons

- Some patterns come with negative consequences (i.e. object proliferation, performance hits, additional layers)
- Consequences are subjective depending on concrete scenarios
- Patterns are subject to different interpretations, misinterpretations, and philosophies
- Patterns can be overused and abused $\rightarrow$ Anti-Patterns

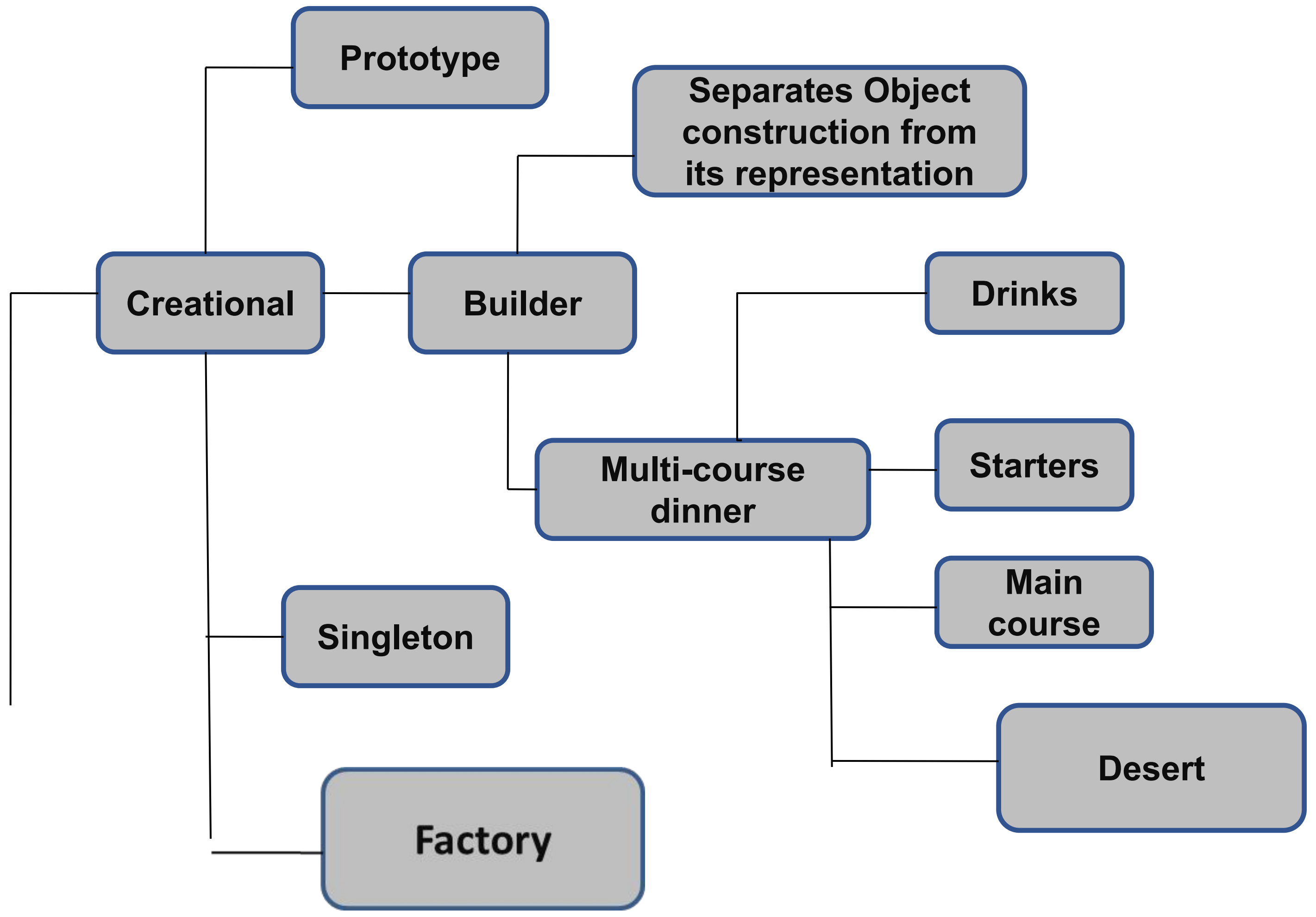
Creational Design Patterns - Types

- Abstract Factory
  - Factory for building related objects
- Builder
  - Separates an object construction from its representation.
- Factory Method
  - Creates an instance of several derived classes.
- Prototype
  - Factory for cloning new instances from a prototype.
- Singleton
  - A class of which a single object can exist.

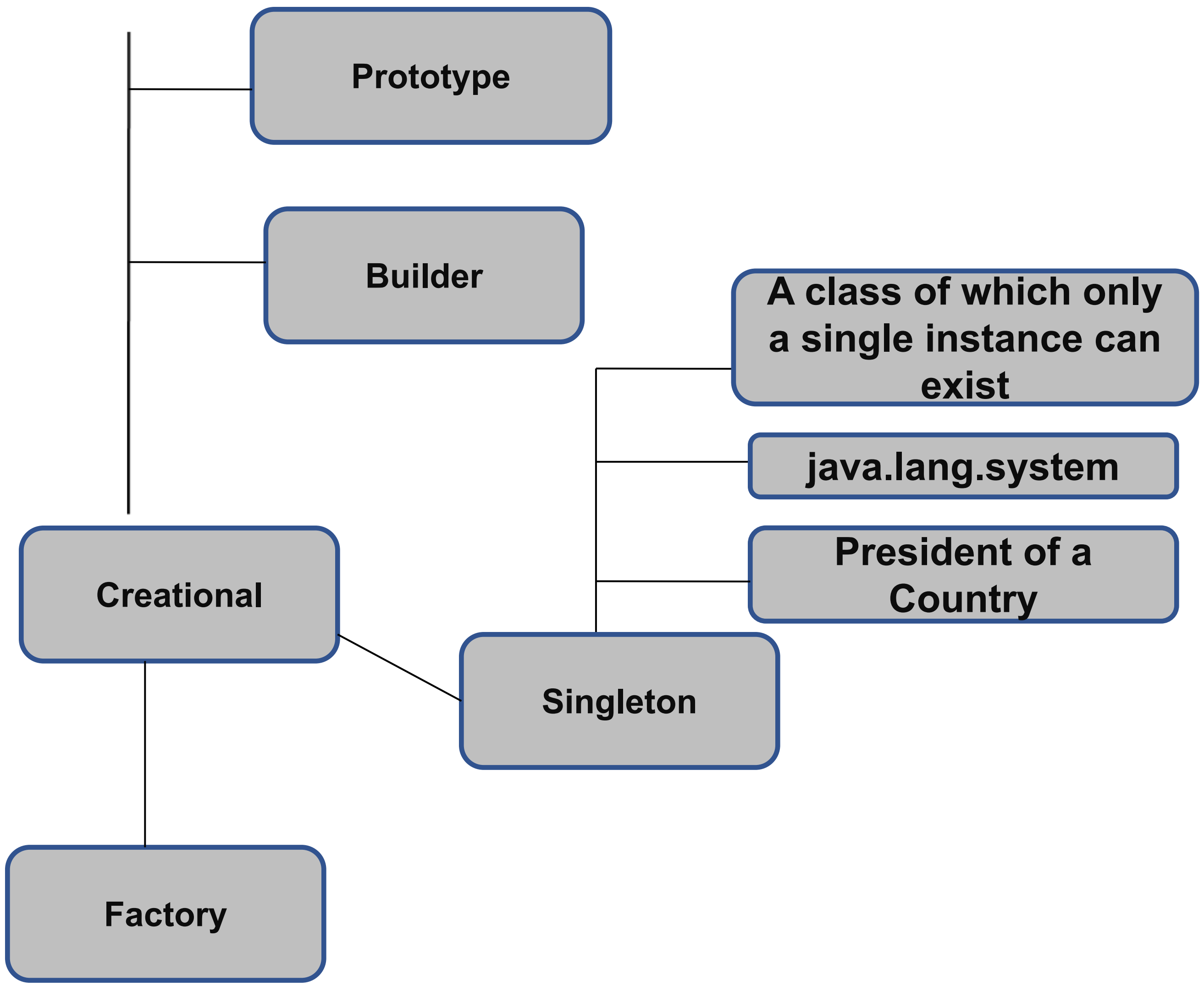
Creational Design Patterns - Types


Creational
Prototype
A fully initialized instance type copied or cloned
Chess Game – Initial set up
Builder
Singleton
Factory

Creational Design Patterns - Types



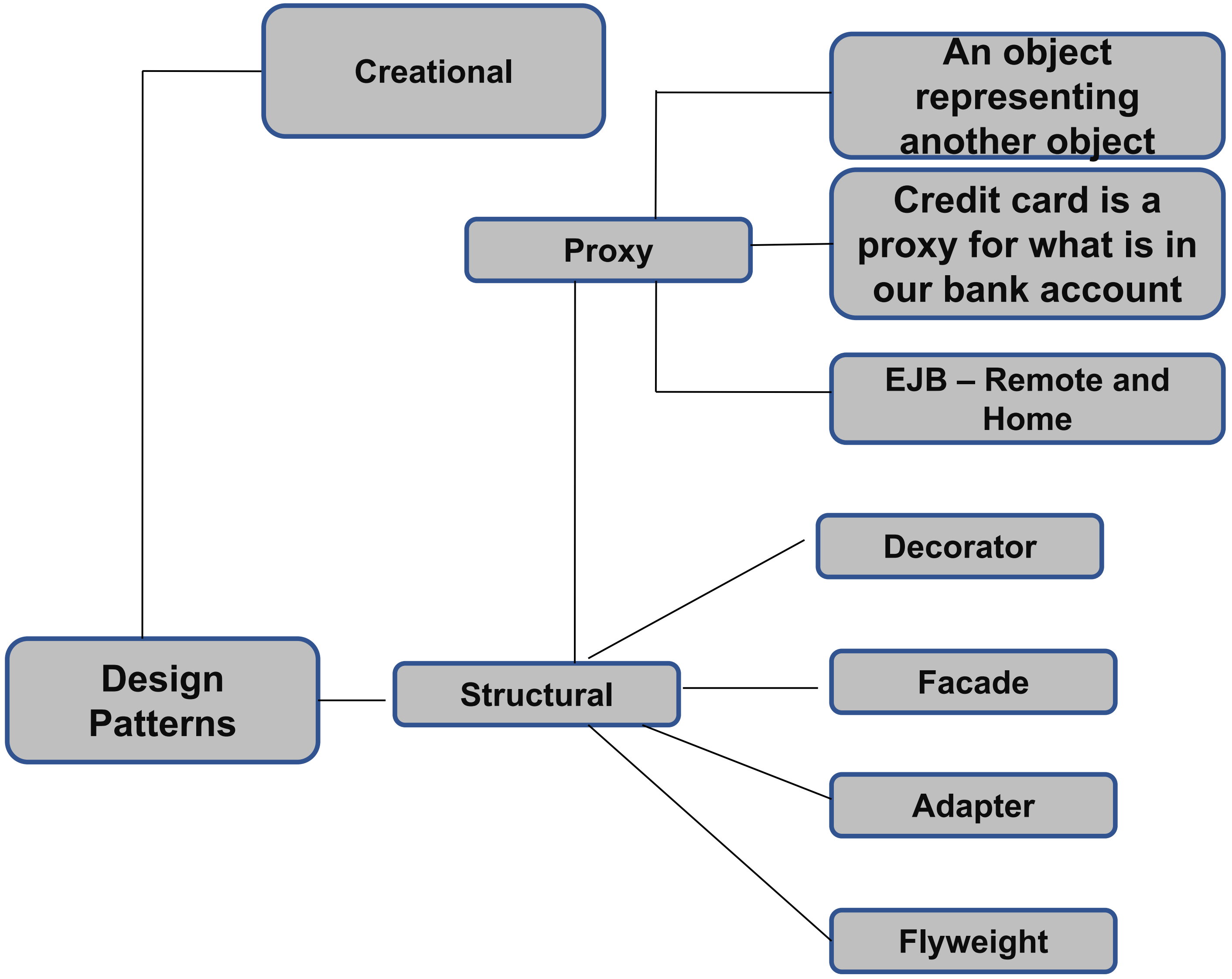
Creational Design Patterns - Types



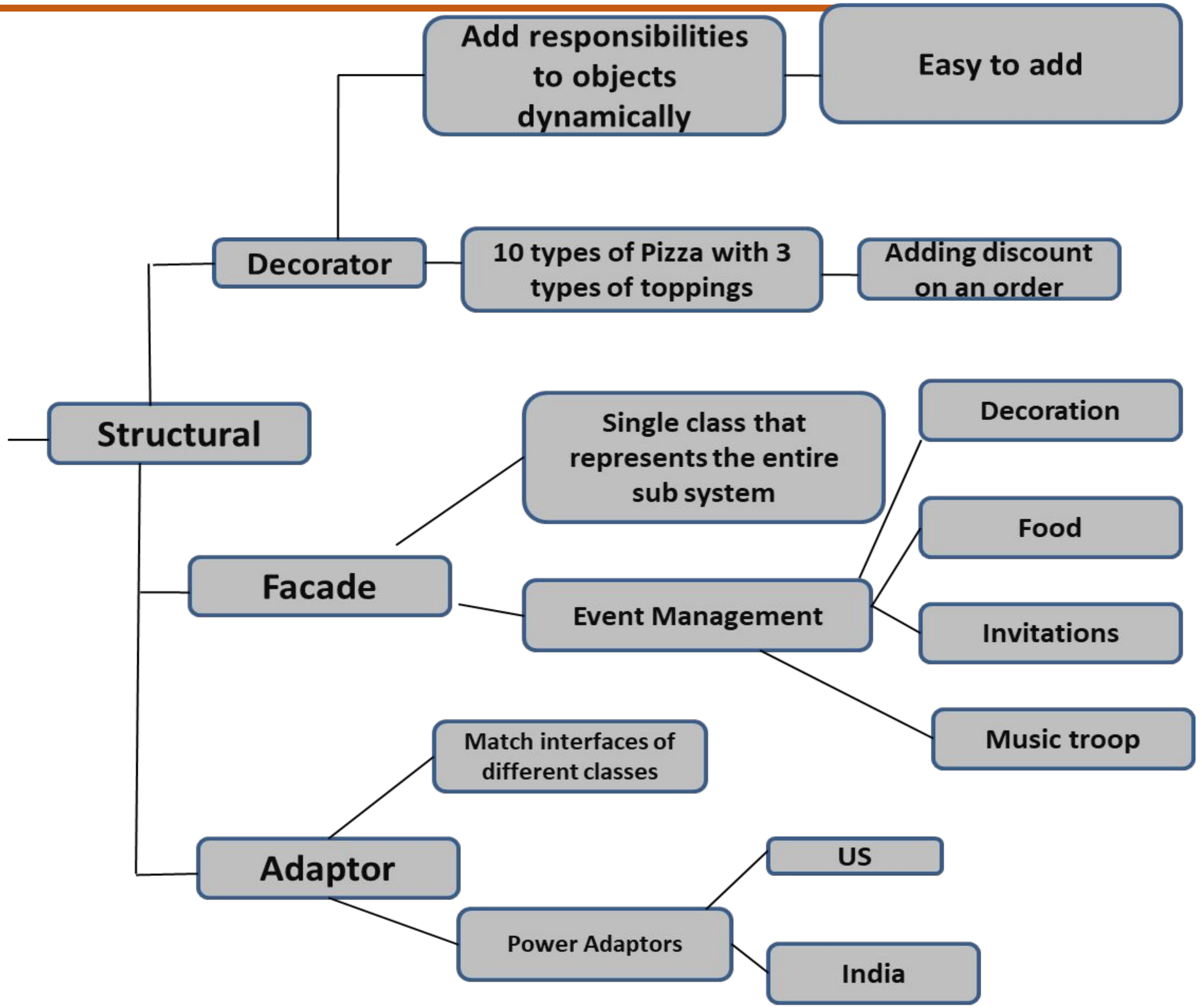
Structural Design Patterns - Types

- Adaptor
  - Match interfaces of different classes.
- Bridge
  - Separates an object’s interface from its implementation.
- Composite
  - A tree structure of simple and composite objects.
- Decorator
  - Add responsibilities of objects dynamically.
- Façade
  - A single class that represents the entire subsystem.
- Flyweight
  - A fine-grained objects shared efficiently.
- Proxy
  - An object representing another object

Structural Design Patterns - Types



Structural Design Patterns - Types


Structural
Decorator
Add responsibilities to objects dynamically
Easy to add
10 types of Pizza with 3 types of toppings
Adding discount on an order
Facade
Single class that represents the entire sub system
Event Management
Decoration
Food
Invitations
Music troop
Adaptor
Match interfaces of different classes
Power Adaptors
US
India

Behavioural Patterns

- Chain of Responsibility
  -Passes a request between a chain of objects
- Command
  - Encapsulate a command request as an object
- Iterator
  - Elements of a collection are accessed sequentially
- Interpreter
  - Language elements are included
- Mediator
  - Defines simplified communication between classes
- Memento
  - Capture and restores an object's internal state
- Observer
  - A way of notifying change to a number of classes
- State
  - Alter an object's behaviour when its state changes
- Strategy
  - Encapsulates an algorithm inside a class
- Template Method
  - Defer the exact steps of an algorithm to a subclass
- Visitor
  - Defines a new operation to a class without change

Creational Patterns – Factory method Pattern

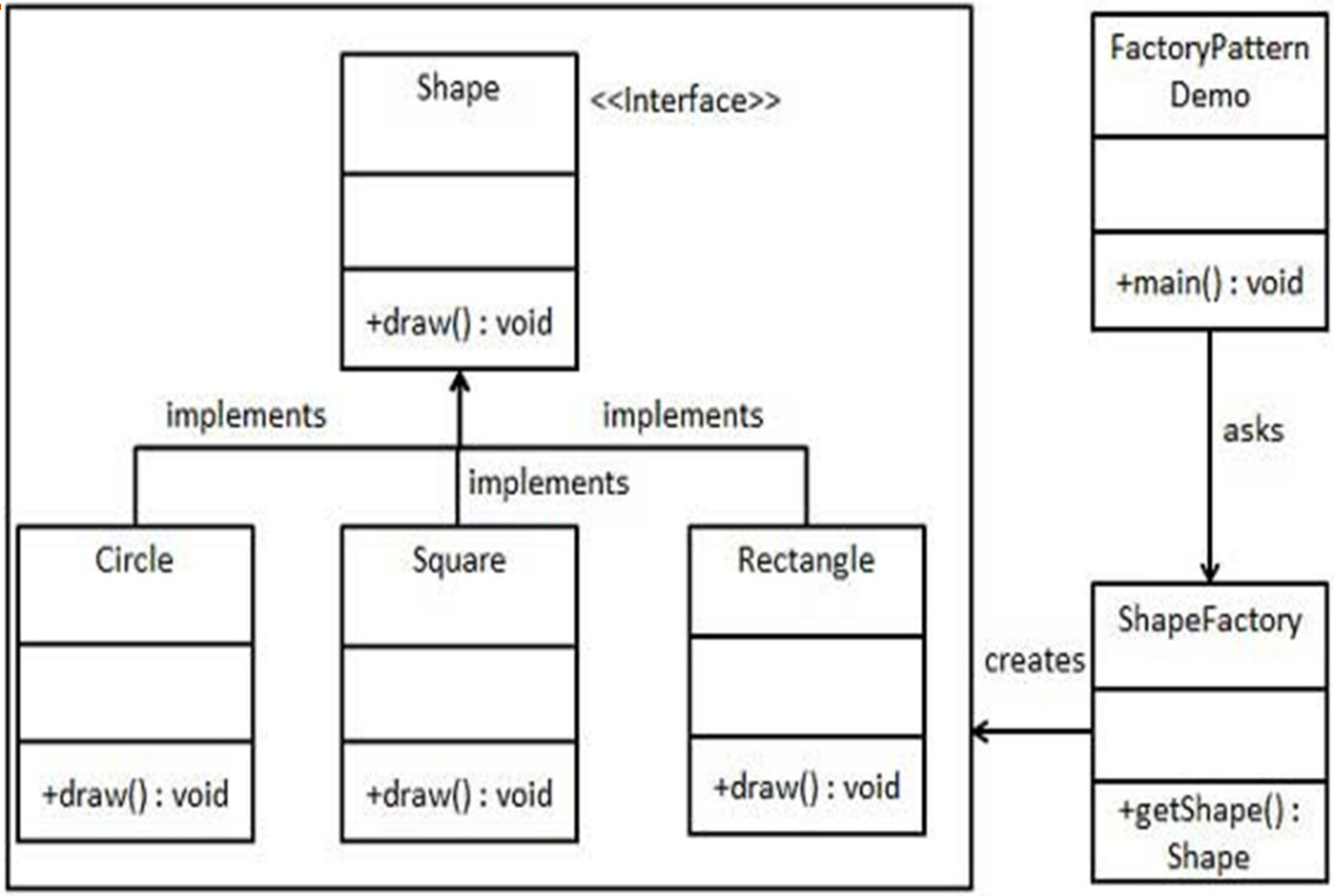
- A factory method is used to create an object without exposing the creation logic to the client .
- The object created will refer the creation class through an interface .

Applicability

Use the Factory Method pattern when,

- a class can´t anticipate the class of objects it must create.
- a class wants its subclasses to specify the objects it creates.
- classes delegate responsibility to one of several helper subclasses, and you want to localize the knowledge of which helper subclass is the delegate.

Creational Patterns – Factory method Pattern



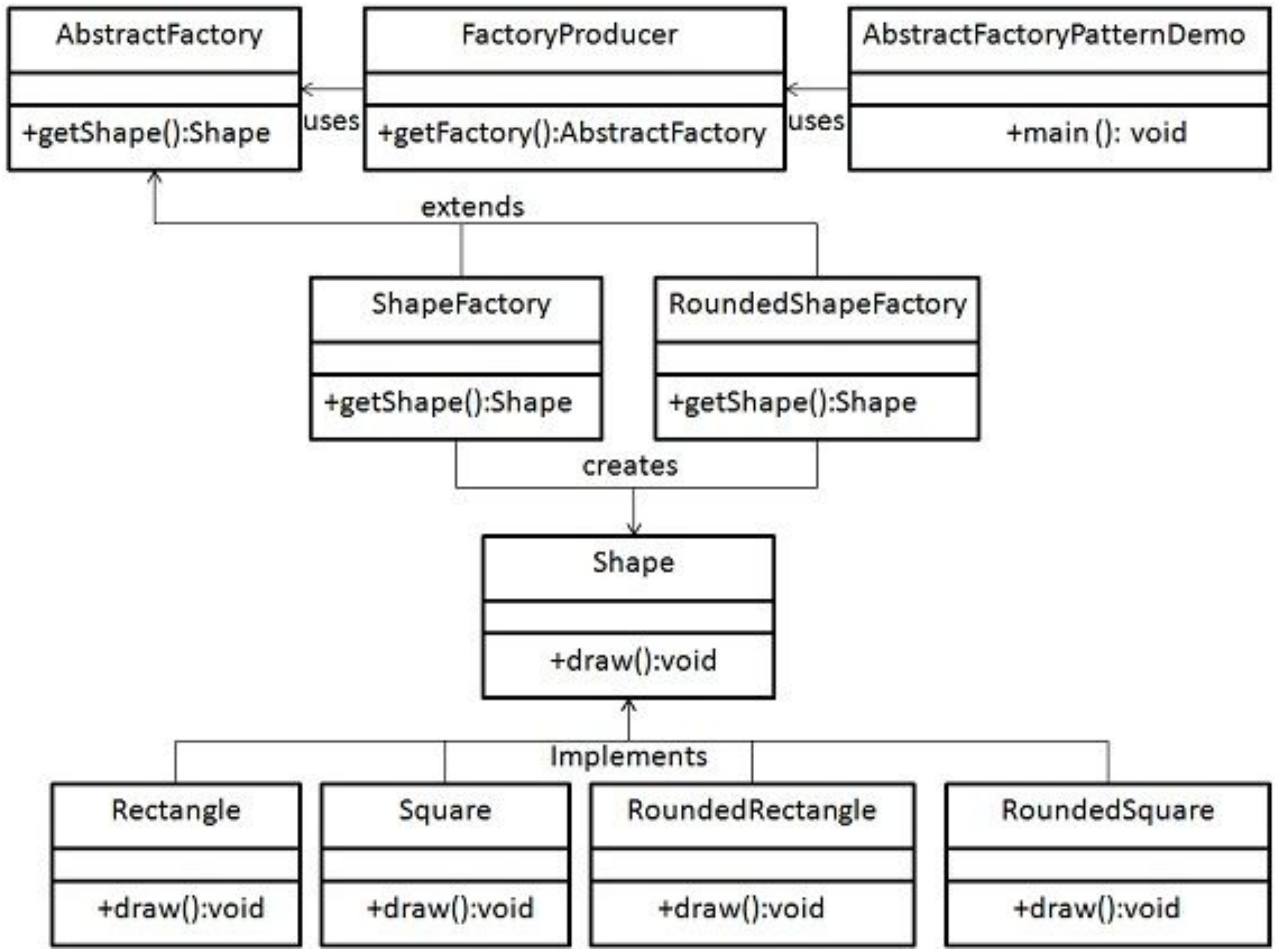
Creational Patterns – Abstract Factory Pattern

Abstract Factory Pattern says that just define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes.

Applicability:

- When the system needs to be independent of how its object are created, composed, and represented.
- When the family of related objects has to be used together, then this constraint needs to be enforced.
- When you want to provide a library of objects that does not show implementations and only reveals interfaces.
- When the system needs to be configured with one of a multiple family of objects.

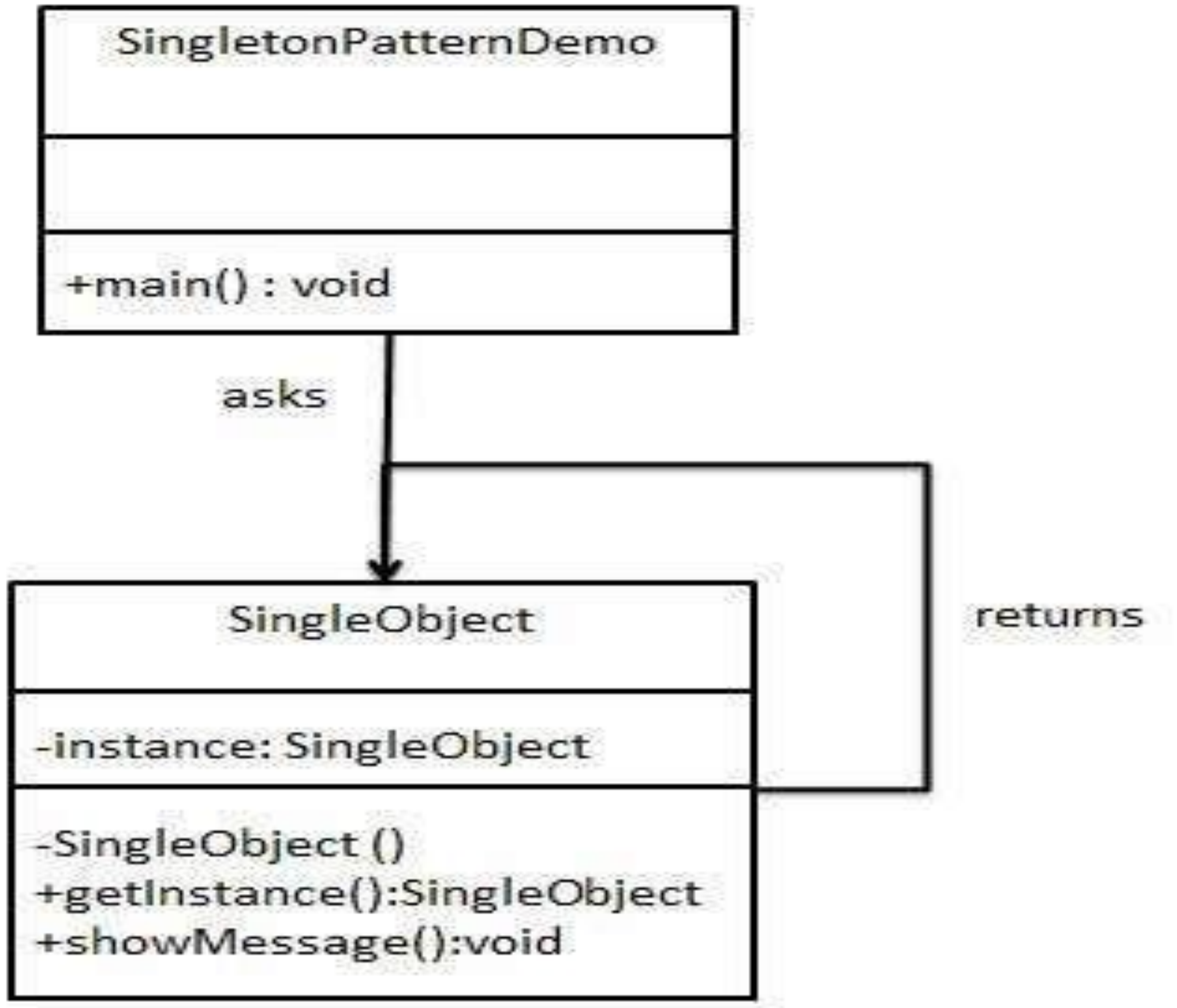
Creational Patterns – Abstract Factory Pattern



Creational Patterns – Singleton Pattern

- Singleton pattern restricts the instantiation of a class and ensures that only one instance of the class exists in the java virtual machine.
- The singleton class must provide a global access point to get the instance of the class.
- Singleton pattern is used for logging, drivers objects, caching and thread pool.
- Singleton design pattern is also used in other design patterns like Abstract Factory, Builder, Prototype, Facade etc.
- Singleton design pattern is used in core java classes also, for example java.lang.Runtime, java.awt.Desktop.

Creational Patterns – Singleton Pattern



Creational Patterns – Builder Pattern

Builder pattern aims to “Separate the construction of a complex object from its representation so that the same construction process can create multiple different representations.”

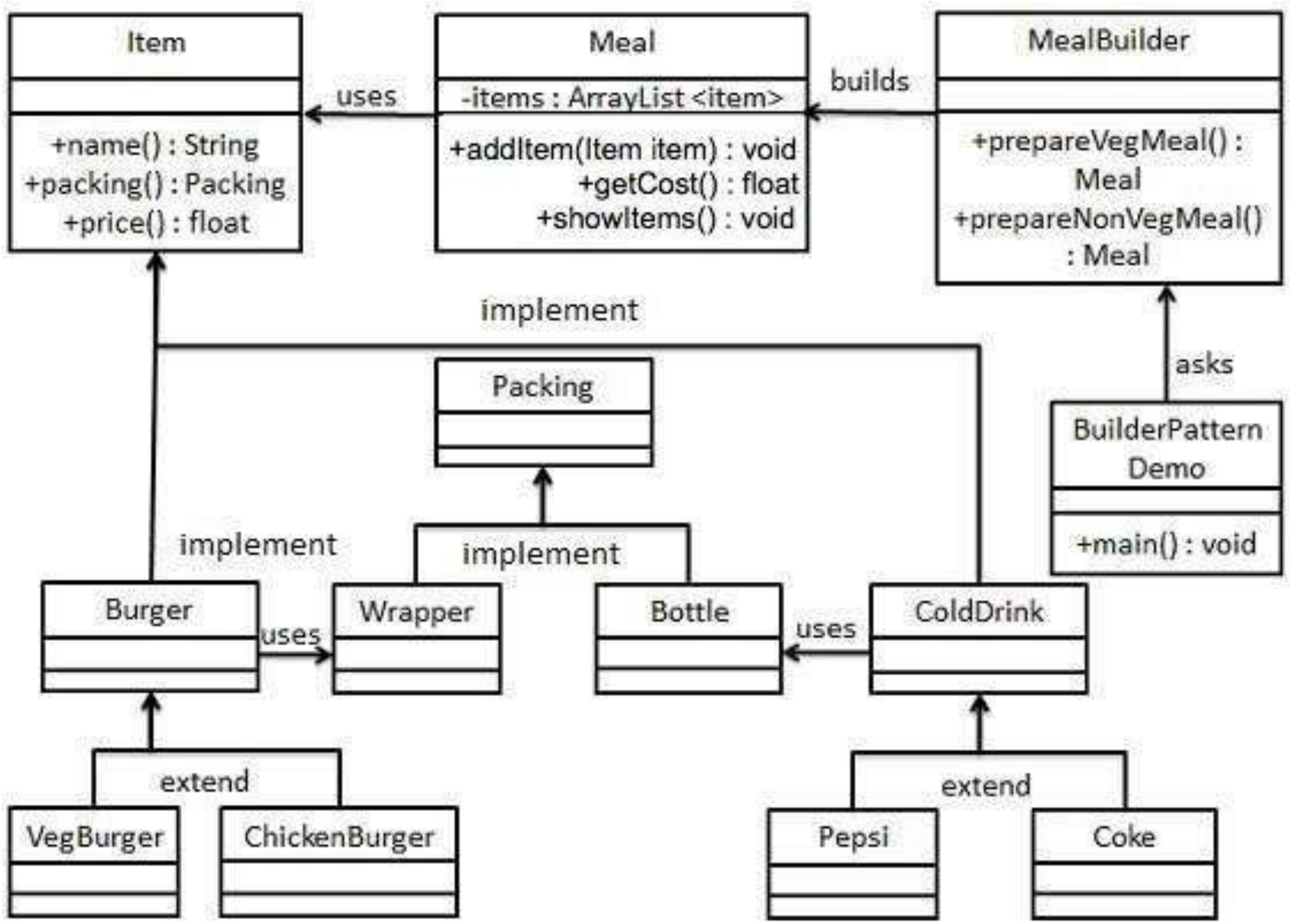
- Builds a complex object using simple objects and using a step by step approach.
- A Builder class builds the final object step by step. This builder is independent of other objects.

Applicability:

- It provides clear separation between the construction and representation of an object.
- It provides better control over construction process.
- It supports to change the internal representation of objects.

Object Oriented Analysis and Design with Java

Creational Patterns – Builder Pattern



References

- https://sourcemaking.com/design_patterns/
- https://refactoring.guru/design-patterns/java

THANK YOU

Department of Computer Science and Engineering