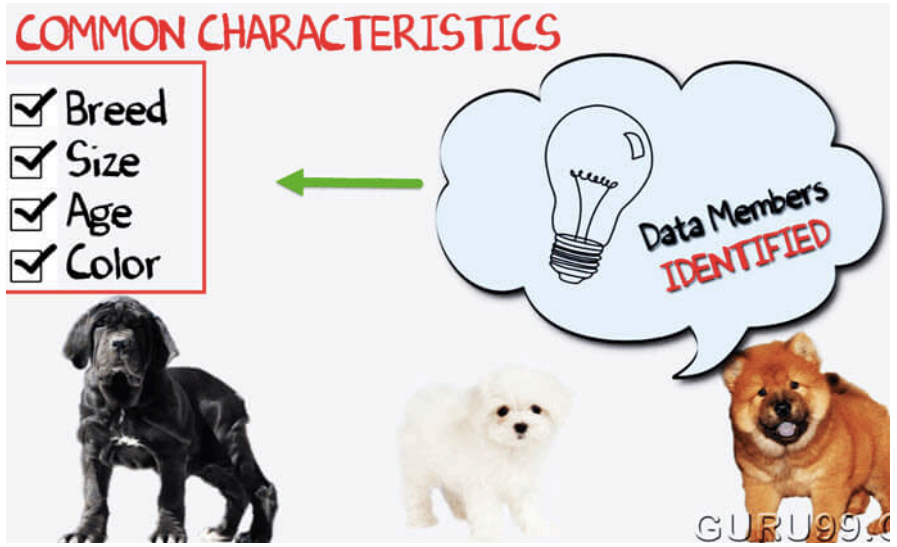
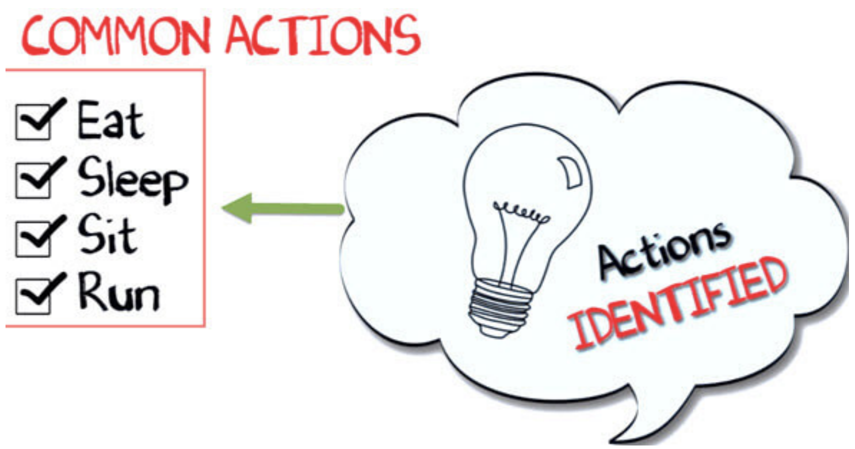
BridgeLabz
Employability Delivered

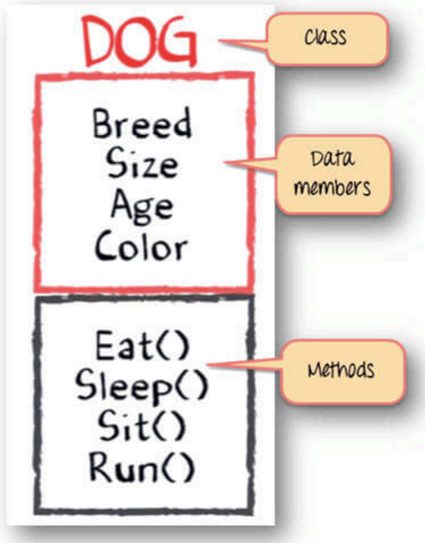
Object Oriented Design Concepts

Classes and Objects

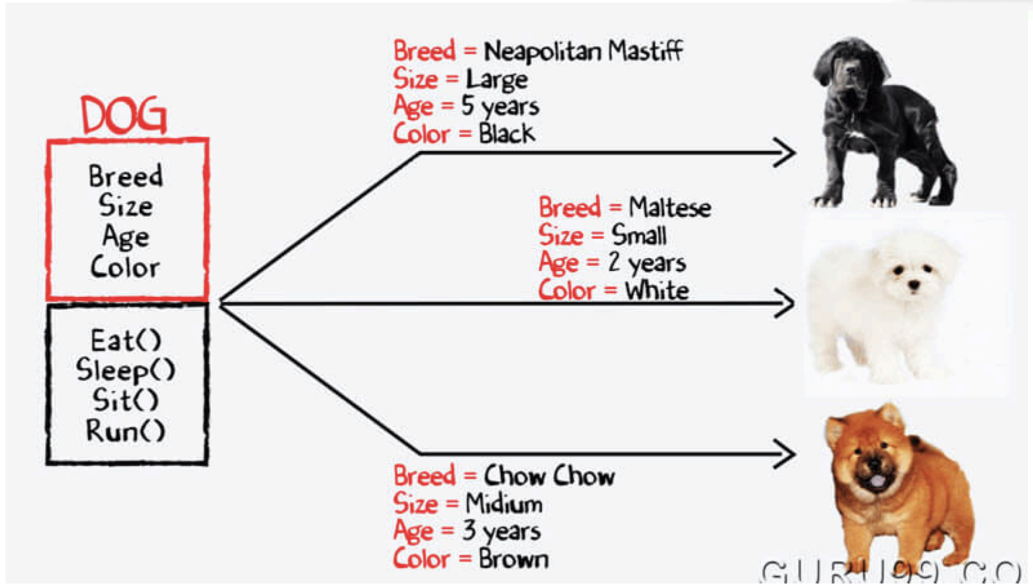
A **class** is a blueprint that defines the variables and the methods (functions) common to all objects of a certain kind.







An **object** is a specimen of a class. Software objects are often used to model real-world objects you find in everyday life. With different values of data members (breed size, age, and color) in Java class, you will get different dog objects.



OOPS Concepts

Abstraction
Abstraction is the concept of hiding the internal details and describing things in simple terms.

Encapsulation
Encapsulation is the technique used to implement abstraction in object oriented programming by using Access modifier private, protected or public keywords

Polymorphism
Polymorphism is the concept where an object behaves differently in different situations. There are two types of polymorphism –
Compile time polymorphism is achieved by method overloading.
Runtime polymorphism is implemented when we have "IS-A" relationship between objects and is achieved by method overriding.

Inheritance
Inheritance is the object oriented programming concept where an object is based on another object.

Association
Association is the OOPS concept to define the relationship between objects. Association defines the multiplicity between objects.

Aggregation
Aggregation is a special type of association. In aggregation, objects have their own life cycle but there is an ownership. Whenever we have "HAS-A" relationship between objects and ownership then it's a case of aggregation.
e.g. A Class has Students

Composition
Composition is a special case of aggregation. Composition is a more restrictive form of aggregation. When the contained object in "HAS-A" relationship can't exist on it's own, then it's a case of composition. There is a **Strong Dependency** between the two objects.
For example, House has-a Room. Here room can't exist without house.

Abstraction

Explain the following:

- Hides the underlying complexity of data
- Helps avoid repetitive code
- Presents only the signature of internal functionality
- Gives flexibility to programmers to change the implementation of the abstract behavior
- Multiple Inheritance can be implemented using ________
- Partial abstraction (0-100%) can be achieved with ________
- Total abstraction (100%) can be achieved with _______
- _______ Relationship is Static in nature
- _______ Relationship is Dynamic in nature

Polymorphism

Explain the following:

- **Compile time polymorphism** is achieved by method overloading.
- **Runtime polymorphism** is achieved by method overriding.

Inheritance

Explain the following:

- A class (child class) can extend another class (parent class) by inheriting its features.
- Implements the DRY (Don’t Repeat Yourself) programming principle.
- Improves code reusability.
- Strong Dependency between Child and Parent Class
- Is Multi-level Inheritance allowed in Java
- Why Multi-type Inheritance not allowed

Polymorphism

Explain the following:

- Static Polymorphism
- Dynamic Polymorphism
- Polymorphism makes it possible to use same Entity in different forms – Explain why????
- Method Overloading is __________ Polymorphism
- Method Overriding is __________ Polymorphism

Association, Aggregation and Composition

Explain the following:

- Establishing a relationship between two unrelated classes, what is this relationship called???
- When you declare two fields of different types in a class, What is this relationship called???
- What relationship is narrower kind of association???
- What occurs when there’s a one-way (HAS-A) relationship between the two classes you associate through their objects??
- What is a stricter form of aggregation???
- What occurs when the two classes you associate are mutually dependent on each other and can’t exist without each other

Association, Aggregation and Composition

Explain the relationship:

- Represents a HAS-A relationship between two classes.
- One-directional
- Two separate classes are associated through their objects.
- Represents a PART-OF relationship between two classes
- A restricted form of aggregation
- Both classes are dependent on each other

Explain the relationship:

- If one class ceases to exist, the other can’t survive alone
- The two classes are unrelated, each can exist without the other one
- Can be a one-to-one, one-to-many, many-to-one, or many-to-many relationship.
- Only one class is dependent on the other.

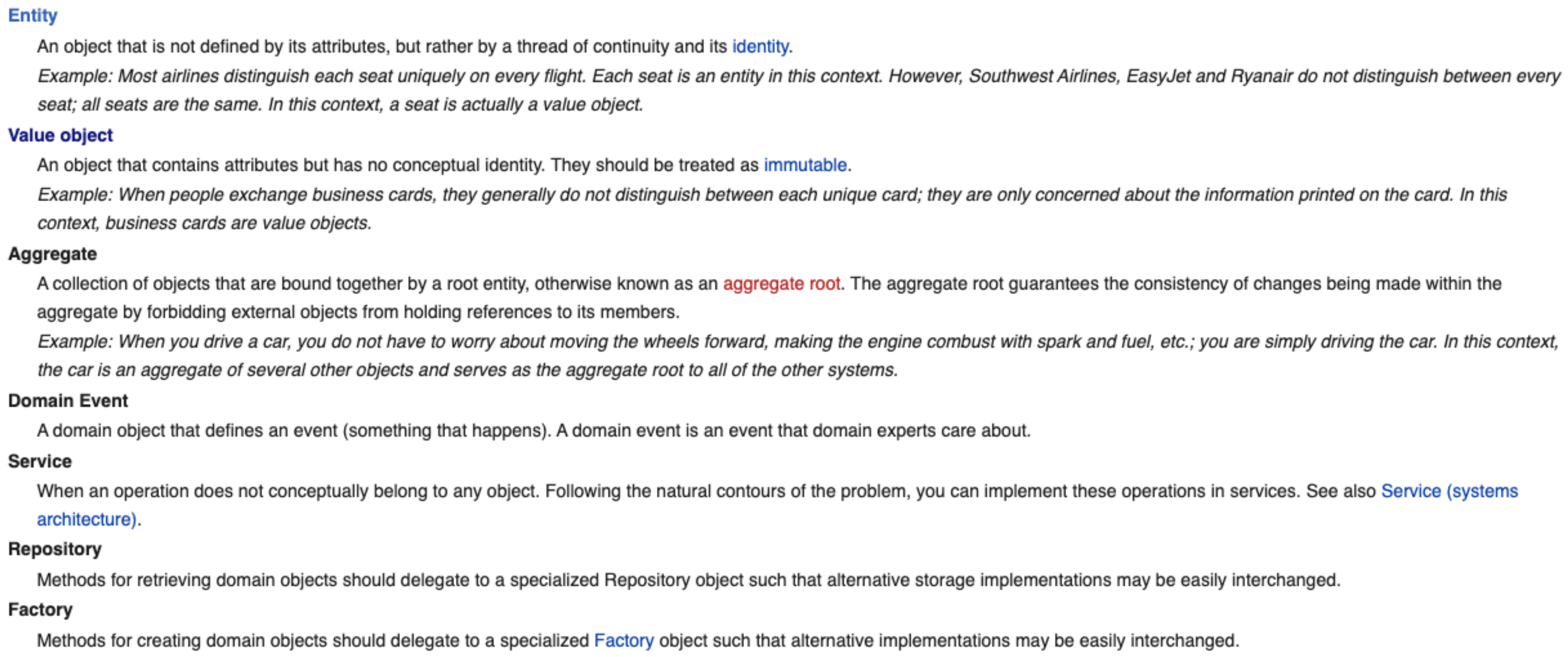
Association, Aggregation and Composition

Example and Explain the following:

- When you declare two fields of different types (e.g. Car and Bicycle) within the same class and make them interact with each other, you have performed ________________. Explain
- When you declare Car and a Passenger, the relationship is called _____________. Explain...
- Directory has Files, this relationship is called ____________
- Car and Engine, this relationship is called ____________

Programming Models

Domain Driven Design (DDD)



Entity

An object that is not defined by its attributes, but rather by a thread of continuity and its identity.

Example: Most airlines distinguish each seat uniquely on every flight. Each seat is an entity in this context. However, Southwest Airlines, EasyJet and Ryanair do not distinguish between every seat; all seats are the same. In this context, a seat is actually a value object.

Value object

An object that contains attributes but has no conceptual identity. They should be treated as immutable.

Example: When people exchange business cards, they generally do not distinguish between each unique card; they are only concerned about the information printed on the card. In this context, business cards are value objects.

Aggregate

A collection of objects that are bound together by a root entity, otherwise known as an aggregate root. The aggregate root guarantees the consistency of changes being made within the aggregate by forbidding external objects from holding references to its members.

Example: When you drive a car, you do not have to worry about moving the wheels forward, making the engine combust with spark and fuel, etc.; you are simply driving the car. In this context, the car is an aggregate of several other objects and serves as the aggregate root to all of the other systems.

Domain Event

A domain object that defines an event (something that happens). A domain event is an event that domain experts care about.

Service

When an operation does not conceptually belong to any object. Following the natural contours of the problem, you can implement these operations in services. See also Service (systems architecture).

Repository

Methods for retrieving domain objects should delegate to a specialized Repository object such that alternative storage implementations may be easily interchanged.

Factory

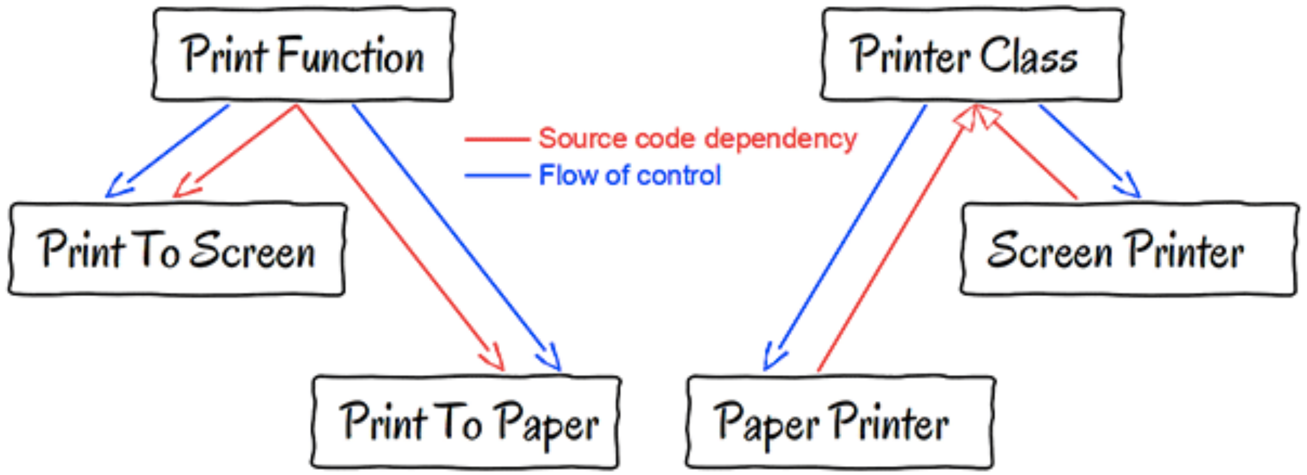
Methods for creating domain objects should delegate to a specialized Factory object such that alternative implementations may be easily interchanged.

Value Objects

- A value object is a small object that represents a simple entity whose equality is not based on identity
- Two value objects are equal when they have the same value, not necessarily being the same object
- Being small, one can have multiple copies of the same value object that represent the same entity
- it is often simpler to create a new object rather than rely on a single instance and use references to it
- Value objects should be immutable: this is required for the implicit contract that two value objects created equal, should remain equal.
- It is also useful for value objects to be immutable, as client code cannot put the value object in an invalid state or introduce buggy behaviour after instantiation.

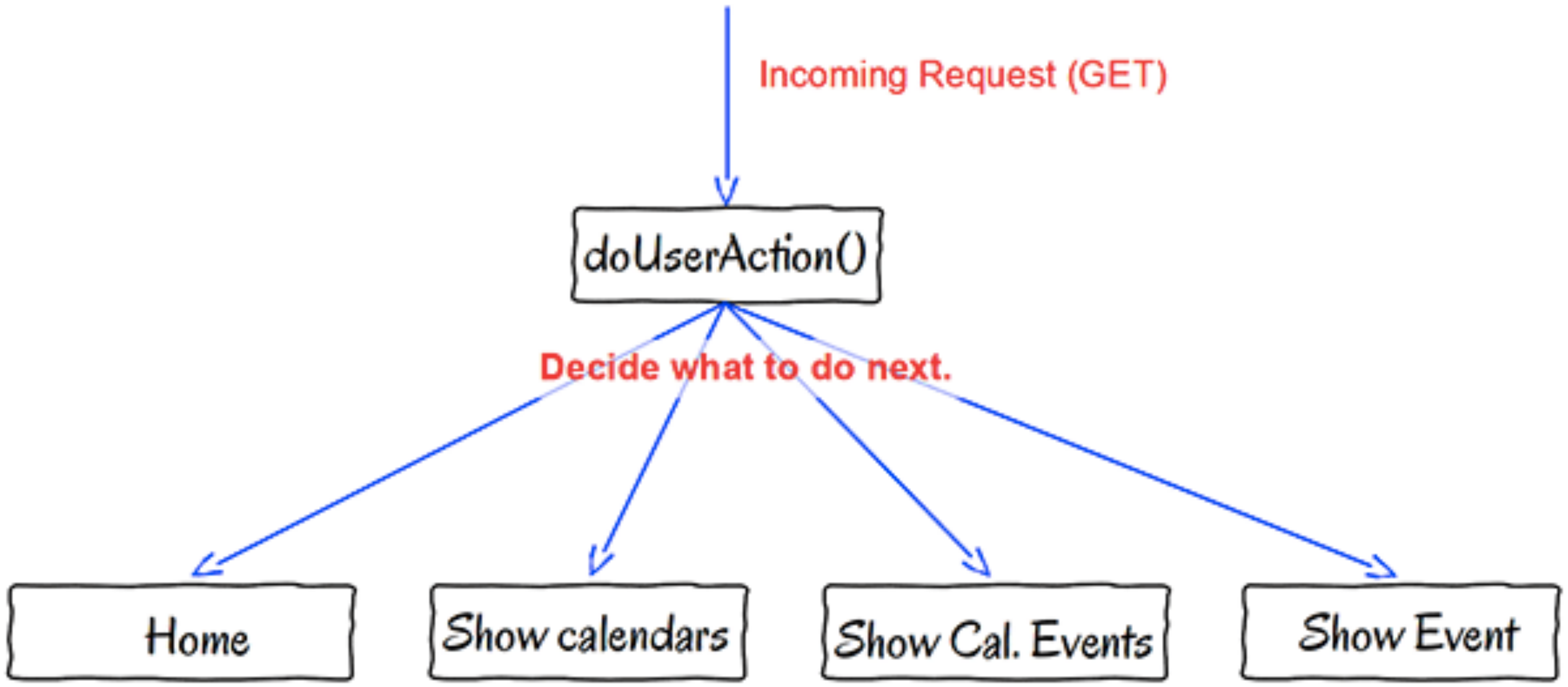
From Procedural to Object Oriented

- Object oriented programming takes away Control flow Statements like Selection Statements and introduces polymorphism.
- OOP can reverse the source code dependency and make it point toward the more abstract implementation, while keeping the flow of control pointing to the more concrete implementation.
- At runtime we want our control to go and reach the most possible concrete and volatile part of our code so that we can get our result exactly as we want.
- In our Source Code we want the concrete and volatile stuff to stay out of the way, to be easy to change and to affect as little as possible of the rest of our code.
- Design Goal - Let the volatile parts change frequently but keep the more abstract parts unmodified.

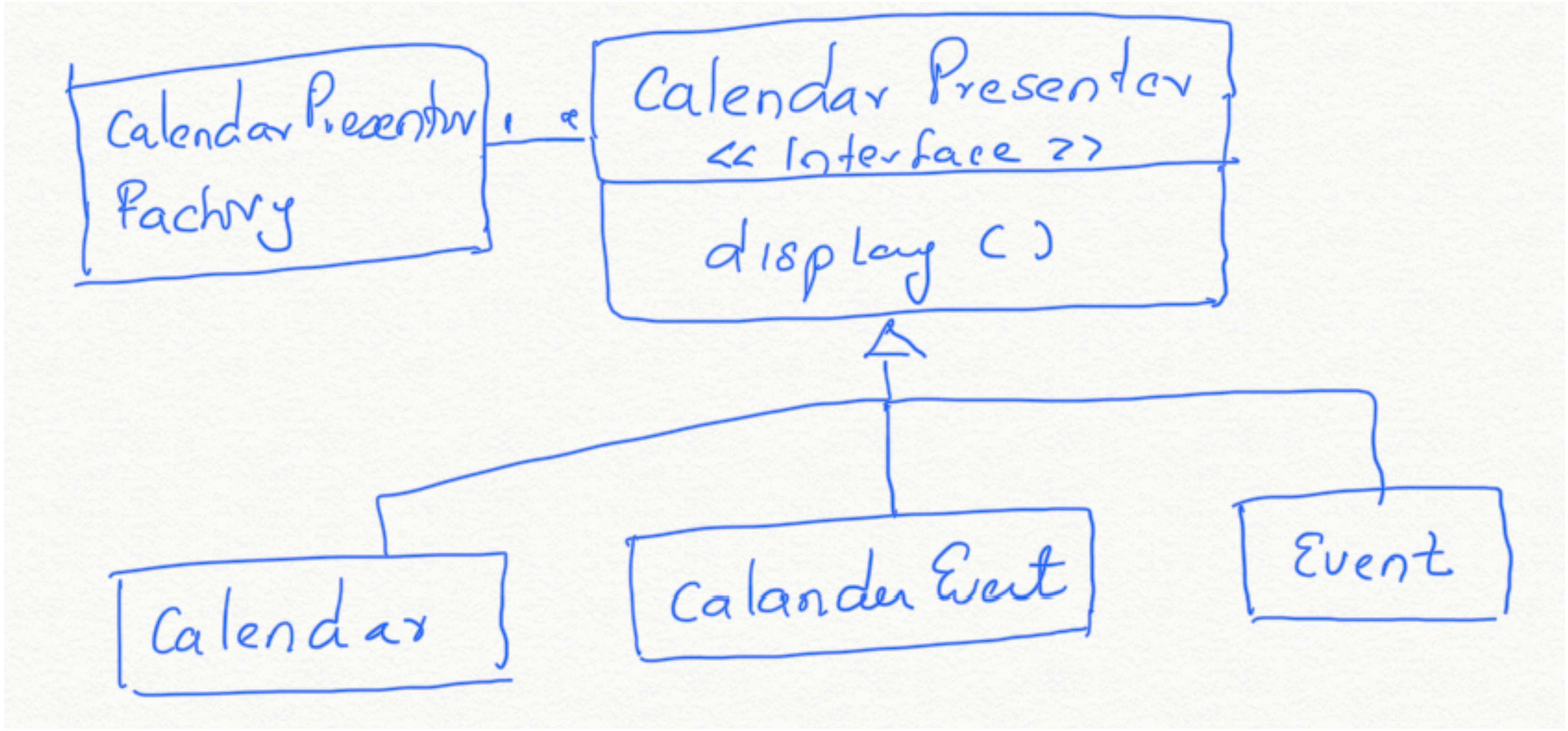


OOP allows extensibility in future we can support File Printer, SMS Printer, WhatsApp Printer, etc.

From Procedural to Object Oriented



From Procedural to Object Oriented


CalendarPresenter Factory
1
*
CalendarPresenter
<< Interface >>
display ()
Calendar
CalanderEvent
Event