**Encapsulation and Data Hiding**

Encapsulation means the bundling of items together into one entity.

Objects in a Java program interact with other objects, by way of their interfaces (list of services (methods)); the data that an object owns and the way that it manages that data are hidden from view (information hiding); only the public services provided by the object are visible on the outside. The data and the way it is managed are said to be encapsulated in the object.

**Objects have *state, behavior,* and *identity***

**Object Identity**: Every object in Java has its own identity. Even if two objects have identical values in their fields, they can be distinguished as different objects.

No matter what its attributes and operations are, an object is always uniquely itself. It retains its identity regardless of changes to its state or behavior.

***Behaviors (*methods**) – the methods in a class are the behaviors that instances of this class can perform on the data.

**State of an object**: the state of an object is the set of values currently stored in its fields. An object's state is defined by the attributes of the object and by the values these have.

**Inheritance (Generalization)**

Object oriented languages support the idea that one type is a "subtype" of another. If class B represents a subtype of class A, this relationship can be realized in the language; class B is said to inherit from class A. Inheritance is achieved using **extends** keyword**.**

A class Subclass inherits from another class Superclass if objects of type Subclass have automatic access to the "available" methods and variables that have been defined in class Superclass.

Generalization groups several classes of objects, which have some properties and operations in

common, to a more general superclass.

**Polymorphism and Late Binding**

**Polymorphism** refers to the ability of an object to take on many forms.

Any Java object that can pass more than one IS-A test is polymorphic.

The good reason for why Polymorphism is need in java is because the concept is extensively used in implementing inheritance.

It plays an important role in allowing objects having different internal structures to share the same external interface.

**Late binding** is referred to as polymorphic references are resolved at run time, this is achieved through method overriding.

Here, Java compiler does not understand which method is called at compilation time. Only JVM decides which method is called at run-time.

**Delegation and Propagation**

Delegation refers to evaluating a member (property or method) of one object (the receiver) in the context of another, original object (the sender). Delegation can be done explicitly, by passing the sending object to the receiving object, which can be done in any object-oriented language; or implicitly, by the member lookup rules of the language, which requires language support for the feature.

**Propagation**

Aggregation and association show significant similarities, among them the ability to combine

multiple instances to a combined instance the dependencies of values along these hierarchies

and their derivation is the subject of **propagation**.

The mechanisms to describe such dependencies and ways to derive values is called propagation.

It supports complex objects which do not own independent data and is based upon the concept

that values are stored only once, i.e., for the properties of the components, and then propagated to the properties of the composite objects. This model guarantees consistency, because the dependent values of the aggregate are derived and need not be updated every time the components are changed.

**CLASS JAVA CODES**

**package** com.mpp.lesson1;

**import** java.util.List;

**public** **class** BackLog {

**private** Long id;

**private** List<Features> features;

}

**package** com.mpp.lesson1;

**public** **class** Developer {

**private** Long id;

**private** String name;

}

**package** com.mpp.lesson1;

**public** **class** Features {

**private** Long id;

**private** String name;

**private** String description;

}

**package** com.mpp.lesson1;

**public** **class** ManagerProject {

**private** Long projectId;

**private** Long managerId;

}

**package** com.mpp.lesson1;

**public** **class** Project {

**private** Long id;

**private** String title;

**private** String description;

}

**package** com.mpp.lesson1;

**public** **class** ProjectManager {

**private** Long id;

**private** String name;

}

**package** com.mpp.lesson1;

**public** **class** Release {

**private** Long id;

**private** String name;

}

**package** com.mpp.lesson1;

**public** **class** ReleaseBackLog {

**private** Long backLogId;

**private** Long releaseId;

**private** String description;

}

package com.mpp.lesson1;

import java.time.LocalDate;

import java.util.List;

public class Sprint {

private Long id;

private String name;

private LocalDate dueDate;

private List<Features> features;

}

**package** com.mpp.lesson1;

**public** **class** SprintDeveloper {

**private** Long sprintId;

**private** Long developerId;

}

**package** com.mpp.lesson1;

**public** **class** SprintRelease {

**private** Long sprintId;

**private** Long releaseId;

**private** **int** noOfSprint;

}