Uttara InfoSolutions Pvt Ltd

www.uttarainfo.com

Object Oriented concepts as discussed in class so far:

Definitions:

1) Abstraction - thought process the designer applies to a given "thing" (object) to hide away unwanted state / behavior and choose only the relevant state/behavior based on the context of the problem. This is the first step in OOAD that is done by a Class Designer for a given problem domain that involves objects.

This is not a feature of the programming language and is not backed by any syntax.

2) Encapsulation - the feature of a programming language (means this is syntactic) using which we can put a boundary for the chosen state variables / behaviors (methods) after applying abstraction.

The ability to create a class is the implementation of this feature in Java.

Advantages:

a) The designer who creates the encapsulation (class) need not be the user of the class and there could be any number of users who use the class. The uses of a Class are a) create object b) subclass c) create reference variables

b) Promotes reuse - The Class User need not have to know the implementation of the class for him to use the class in his programs (just having javadocs is enough for him to use the class and he not have access to the code as well).

c) Promotes easy maintenance - The implementation of the class (given by method body + state) is hidden from the user of the class. Hence if a bug needs to be fixed, the bug fix needs to be done in only one place as any change in implementation affects all the objects of this class used in every program.

d) creates a specialist - Every class creates a specialist object design with related data and functions. That means it's easy to build such specialists and piece together a solution using many of such specialists to form a complex OO solution. This helps in constructing solutions bottom up.

3) Data hiding:

The feature of the language using which the designer can choose to hide state from direct access to the user of the class. Since state impacts behavior, it's very important that state changes be controlled as per problem domain and hence only the designers code must modify state (hence setters/getters are exposed for user to access state indirectly).

Advantages:

a) Ensures robustness/reliability of objects: since the user of the class cannot modify the state in uncontrolled manner, the working of the object is as per the implementation given by the designer and hence objects conform to how they are expected to work at all times. This reduces bugs/defects thereby increasing robustness of the code.

4) Inheritance:

The feature of the programming language using which entities can be linked to create the IS-A relationship. Java provides

class and interface inheritance as multilevel and both multiple & multilevel inheritance respectively. Once you link 2 classes with IS-A (extends keyword in header of class):

1) Every subclass object will/must behave like a parent (means whatever you can ask a parent to do, a subclass object must be able to do as well - contract of inheritance)

2) Parent ref can be used to point to subtype objects (and then you can only invoke parent methods)

3) Creating an object of sublcass involves creating all parents state in parent layers (via constructor chaining)

4) Override methods inherited from parent

5) Add delta functionality

Overriding: Feature of the language for a subclasser to modify the implementation of a method that is inherited from the parent by redefining the method header and providing a new implementation - done only when problem domain requires so - to provide the enhancement.

DMD - Dynamic Method Dispatch / Dynamic linking / runtime linking -
Ability of JVM to link a method implementation based on type of object (ref var has no role to play in this)

LSP - Lishkov Substitution Principle - Ability to use a parent ref to point to a subtype object (works cuz as per the inheritance contract, whatever a parent can do, every of its subtype can do and using parent ref, you can only invoke parent specific methods - checked by compiler).

Polymorphism:
Feature(s) of the language. In Java - given to us as static polymorphism (compile time), dynamic poly (runtime) and parametric polymorphism.

Static poly: Feature to overload methods / contstructors / operators (developer cannot overload operators explicitly in Java but + is implicitly overloaded).
How to overload method/constructor?

By changing argument list (either order, number or datatype of the args)

Use:
Constructor overloading - To provide convenience to the user to be able to construct objects by passing different arg lists.
Method overloading - To provide convenience to the user of the class to be able to invoke different implementations but with same name, again depending on the arg list passed.

Runtime poly: Feature of JVM to pick one method implementation from the several that might exist in a class hierarchy (due to overriding).

Use:
Runtime poly is useful when we use it by coding for it - that is only if we code to the parent ref! If we do, then we create flexible, generic, loosely coupled code which can use enhanced implementations that exists now and that might exist in the future.

Inheritance + overriding + DMD + LSP = Polymorphic code!

Polymorphic code is created when we code to the parent ref:

a) Create a parameter ref var of parent type

b) Return data type of a method is of parent type

c) Instance variable is of parent type

d) Local variable is of parent type