Object Oriented Development with Java

(CT038-3-2 and Version VC1)

Encapsulation and Polymorphism

Object Oriented Programming

Topic & Structure of the lesson

- Polymorphism
- Encapsulation

Learning outcomes

- At the end of this lecture you should be able to:
  - Understand the implementation of polymorphism
  - Understand the implementation of encapsulations

Key terms you must be able to use

If you have mastered this topic, you should be able to use the following terms correctly in your assessments:

- Overloading
- Overriding
- Getter
- setter

Polymorphism

Polymorphism:- It is the ability of an object to take on many forms. In java language, polymorphism is essentially considered into two versions.

- Compile time polymorphism (method overloading/ static binding)
- Runtime polymorphism (method overriding/ dynamic binding)

Compile time polymorphism (method overloading):

- This is used to write the program in such a way, that flow of control is decided in compile time itself.
- It is achieved using method overloading(implicitly).
- In method overloading, an object can have two or more methods with same name but with their method parameters different..

- These parameters may be different on two bases:
- Parameter type: Type of method parameters can be different.

```
public void num(double a, double b){..}
public void num (float a, float b){..}
public void num (int a, int b){..}
```

- Parameter count: Functions accepting different number of parameters.

EmployeeFactory.create(String firstName, String lastName){...}

EmployeeFactory.create(Integer id, String firstName, String lastName

Runtime polymorphism (method overriding):-

- Feature that allows a subclass or child class to provide a specific implementation of a method that is already provided by one of its super classes or parent classes.
  public class students { public void study(){
  System.out.println(“OOP"); } }

```
class Ghazanfar extends students{
    public void study(){
System.out.println(“OOP,Discrete"); } }
class Rafah extends students{
    public void study(){
    System.out.println(“Report writing"); } }
```

Now which study() method will be called?

- Depends on type of actual instance created on runtime.

```
public class Demo {
    public static void main(String[] args) {
students a1 = new Ghazanfar();
 a1.study();
students a2 = new rafah();
a2.study(); } }
```

Encapsulation

- Encapsulation is the technique of making the fields in a class private and providing access to the fields via public methods.
- Encapsulation also can be described as a protective barrier that prevents the code and data being randomly accessed by other code defined outside the class.

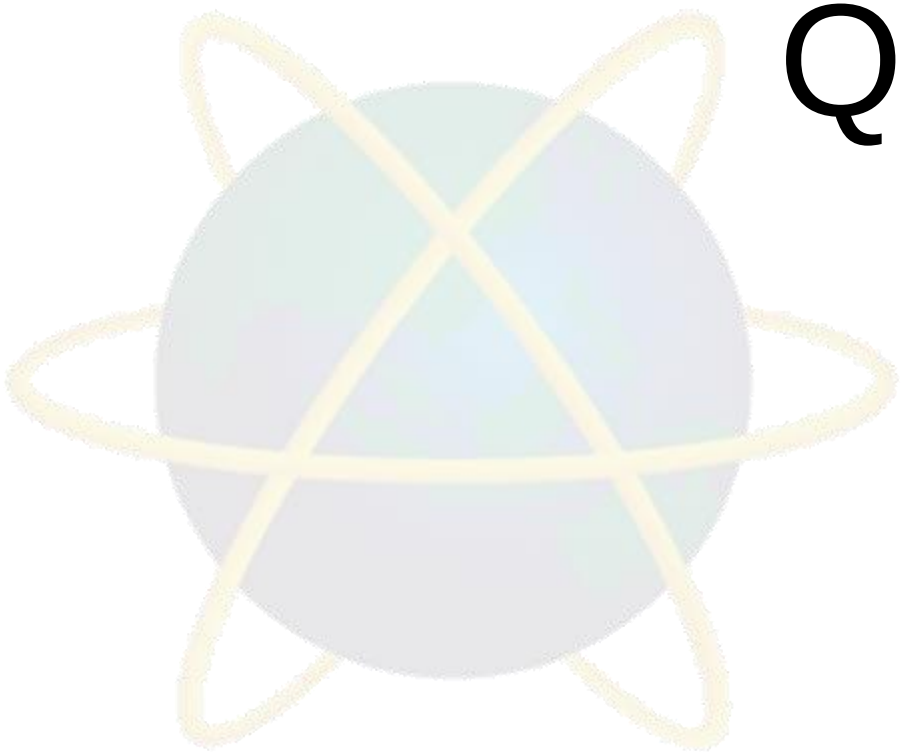
- To hide the internal implementation details of the class
- Can safely modified the implementation without worrying breaking the existing code that uses the class
- Protect class against accidental/ willful stupidity
- Keeps class tidy by keeping the visible fields to a minimum
- Easier to use and understand

Quick Review Questions

- What is an overriding?
- What is an overloading?
- Difference between overloading and overriding
- encapsulation

Summary of Main Teaching Points

- Overloading
- Overriding
- Encapsulation



Q & A

Next Session

A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

-Introduction
-Using Packages
-Accessing Packages
-Packages Naming Conventions
-Package Declaration
-Adding Class to a Package