PREGRADO

UNIDAD 1 | OVERVIEW

JAVA OO FEATURES

SI705 | Arquitectura de Aplicaciones Web

Al finalizar la unidad, el estudiante desarrolla aplicaciones web básicas en un ambiente de desarrollo ágil.

AGENDA

INTRO
ABSTRACTION
ENCAPSULATION
INHERITANCE
POLYMORPHISM

AGENDA

INTRO
ABSTRACTION
ENCAPSULATION
INHERITANCE
POLYMORPHISM

Abstraction

Simplificar conceptos en comparación con algo similar en el mundo real, ocultar información que no es relevante para el contexto, mostrando solo información importante.

Dos maneras:

- *Data abstraction*
- *Control abstraction*

Data Abstraction

Crear tipos de dato complejos en base a tipos más pequeños.

```
public class Employee {
    private Department department;
    private Address address;
    private Education education;
    //So on...
}
```

Control Abstraction

Ocultar la secuencia de acciones de una tarea compleja.

Cambios posteriores en la lógica no afectan al cliente.

```java
public class EmployeeManager {
    public Address getPrefferedAddress(Employee e){
        //Get all addresses from database
        //Apply logic to determine which address is preferred
        //Return address
    }
}
```

AGENDA

INTRO
ABSTRACTION
ENCAPSULATION
INHERITANCE
POLYMORPHISM

Encapsulation

Envolver datos y métodos dentro de clases, a la vez que ocultar implementación.

Abarca:

- *Information hiding*
- *Implementation hiding*

Information hiding

*Se logra en base a modificadores de control de acceso (***public***,* ***private***, ***protected****).*

```java
class InformationHiding
{
    //Restrict direct access to inward data
    private ArrayList items = new ArrayList();

    //Provide a way to access data – internal logic can safely be changed in future
    public ArrayList getItems(){
        return items;
    }
}
```

Implementation hiding

Permite modificar cómo es cumplida la responsabilidad de un objeto, en escenarios con diseño o requisitos cambiantes.

```java
interface ImplementationHiding {
    Integer sumAllItems(ArrayList items);
}

class InformationHiding implements ImplementationHiding
{
    //Restrict direct access to inward data
    private ArrayList items = new ArrayList();

    //Provide a way to access data – internal logic can safely be changed in future
    public ArrayList getItems(){
        return items;
    }

    public Integer sumAllItems(ArrayList items) {
        //You can change the sequence or even whole implementation logic
        //without affecting the client
    }
}
```

AGENDA

INTRO
ABSTRACTION
ENCAPSULATION
INHERITANCE
POLYMORPHISM

Inheritance

*Establece relaciones parent-child. Utiliza **extends** keyword.*

```
class Employee
{
    private Department department;
    private Address address;
    private Education education;
    //So on...
}
class Manager extends Employee {
    private List<Employee> reportees;
}
```

AGENDA

INTRO
ABSTRACTION
ENCAPSULATION
INHERITANCE
POLYMORPHISM

Polymorphism

Crear funciones o variables de referencia que se comportan diferente en distintos contextos de programación.

```
Object o = new Object(); //o can hold the reference of any subtype
Object o = new String(); //String is a subclass of Object class
Object o = new Integer();//Integer is a subclass of Object class
```

Dos versiones:

- *Compile time polymorphism.*
- *Runtime polymorphism.*

Compile Time Polymorphism

Static binding o method overloading.

```
public static double Math.max(double a, double b){..}
public static float Math.max(float a, float b){..}
public static int Math.max(int a, int b){..}
public static long Math.max(long a, long b){..}
```

```
EmployeeFactory.create(String firstName, String lastName){...}
EmployeeFactory.create(Integer id, String firstName, String lastName){...}
```

Runtime Polymorphism

Dynamic binding o method overriding.

```
public class Animal {
    public void makeNoise() {
        System.out.println("Some sound");
    }
}

class Dog extends Animal {
    public void makeNoise() {
        System.out.println("Bark");
    }
}

class Cat extends Animal {
    public void makeNoise() {
        System.out.println("Meawoo");
    }
}
```

```
public class Demo {
    public static void main(String[] args) {
        Animal a1 = new Cat();
        a1.makeNoise(); //Prints Meowoo

        Animal a2 = new Dog();
        a2.makeNoise(); //Prints Bark
    }
}
```

Method Overloading Rules

Cambiar firma del método: Número de argumentos, tipo de argumentos, orden de argumentos.

Tipo de retorno no es parte de la firma: Cambiar tipo de retorno no se considera overloading.

Method Overloading Rules

Lanzar (Thrown) excepciones no se considera overloading: Si lanza la misma excepción, otra o no lanza excepción, no afecta carga del método.

```
public class DemoClass {
    // Overloaded method
    public Integer sum(Integer a, Integer b) throws NullPointerException {
        return a + b;
    }

    // Overloading method
    //Not valid; Compile time error
    public Integer sum(Integer a, Integer b) throws Exception {
         return null;
    }
}
```

Method Overriding Rules

Lista de argumentos en overriden y overriding methods debe ser exáctamente la misma: De otro modo es overloading.

Tipo de retorno en overriding method debe ser igual al de overriden method.

```java
public class SuperClass {
    //Overriden method
    public Number sum(Integer a, Integer b) {
        return a + b;
    }
}

class SubClass extends SuperClass {
    //Overriding method
    //Integer extends Number; so it's valid
    @Override
    public Integer sum(Integer a, Integer b) {
        return a + b;
    }
}
```

Method Overriding Rules

Override no aplica a private, final o static methods.

```
public class SuperClass {
    //private method; overriding not possible
    private Integer sum(Integer a, Integer b) {
        return a + b;
    }
}

class SubClass extends SuperClass {
    //Overriding method
    public Integer sum(Integer a, Integer b) {
        return a + b;
    }
}
```

Method Overriding Rules

Overriding method no puede lanzar checked Exception de jerarquía mayor de la que se lanza en overriden method.

```java
public class SuperClass {
    //Overriden method
    public Integer sum(Integer a, Integer b) throws FileNotFoundException {
        return a + b;
    }
}

class SubClass extends SuperClass {
    //Overriding method
    //Not valid; Compile time error
    public Integer sum(Integer a, Integer b) throws IOException {
        return a + b;
    }
    //Exception IOException is not compatible with throws clause
    //in SuperClass.sum(Integer, Integer)
    //It's valid; Don't declare the exception at all is permitted.
    public Integer sum(Integer a, Integer b)  {
        return a + b;
    }
}
```

Method Overriding Rules

Overriding method no puede reducir access scope en overriden method.

```
public class SuperClass {
    //Overriden method
    protected Integer sum(Integer a, Integer b) {
        return a + b;
    }
}

class SubClass extends SuperClass {
    //Overriding method
    //Not valid; Compile time error:
    //Cannot reduce the visibility of the inherited method from SuperClass
    private Integer sum(Integer a, Integer b)  {
        return a + b;
    }
}
```

@Overriding annotation

Verifica que se cumplan todas las reglas de overriding. Cualquier issue provoca un error de compilación.

RESUMEN

Recordemos

Java es un lenguaje de programación que soporta el paradigma orientado a objetos.

Implementa abstracción, encapsulamiento, herencia, polimorfismo.

REFERENCIAS

Para profundizar

https://docs.oracle.com/javase/tutorial/java/concepts/index.html

https://www.w3schools.com/java/java_oop.asp

PREGRADO

Ingeniería de Sistemas de Información

Escuela de Ingeniería de Sistemas y Computación | Facultad de Ingeniería

UPC
*Universidad Peruana
de Ciencias Aplicadas*

*Prolongación Primavera 2390,
Monterrico, Santiago de Surco
Lima 33 - Perú
T 511 313 3333
https://www.upc.edu.pe*

exígete, innova