UNI P ACADEMY OOP NOTES

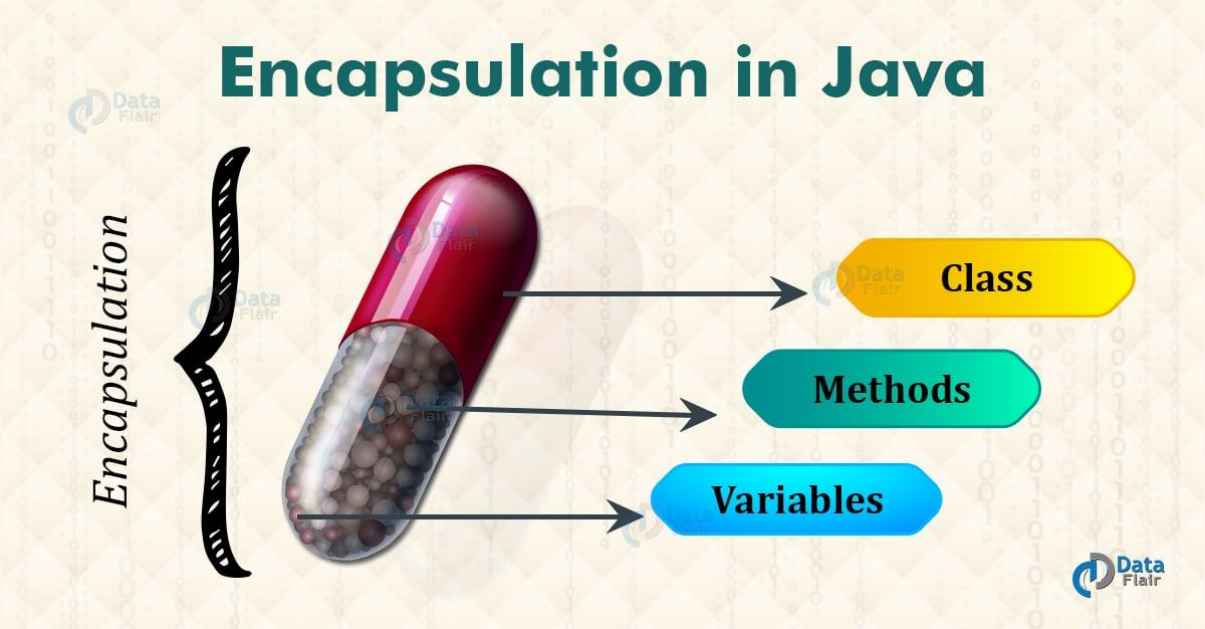
Definition of OOP: its programming paradigm that focuses on the concepts of objects

FOUR OOP PRINCIPLES

1. ENCAPSULATION
2. ABSTRUCTION
3. INHERITANCE
4. POLYMORPHISM

ENCAPLUSTION

DEFINITION: An oop principle that use the mechanism of wrapping data and methods that act on that data into a single unit which can be an object.



Our main Aim on encapsulation is data hiding **what do I mean?**

It means hiding the internal state of the object from the outside world mainly to prevent unwanted changes, for example based on the picture provided we are using a capsule to hide wants inside and we don't want external forces to change the context of what's inside.

TWO THINGS WE TAKE NOTE

1. ACESS SPECIFIER (private)
2. TWO PUBLIC METHODS
   -A SETTER METHODS: ALLOWS ACCESS TO OUR OBJECT/CAPSULE
   -A GETTER METHODS: ALLOWS TO RETRIEVE THE CONTEXT OF THE CAPSULE/ OBJEC

CODE SNIPET:

```
class Car {

   private int speed; // private variable

   // public method to set the value of speed

   public void setSpeed(int speed) {// our setter method

      if (speed > 0) {

         this.speed = speed; // (this.) used to indicate that anywhere in our code this is the speed value we will be using

      }

   }

   // public method to get the value of speed

   public int getSpeed() {// our getter method

      return speed;

   }

}
```

:: We have a variable speed which we made private indicating that we are using encapsulation and no direct access is allowed from external entry.

-Setter method we use it to access the variable speed and initialize it to a value

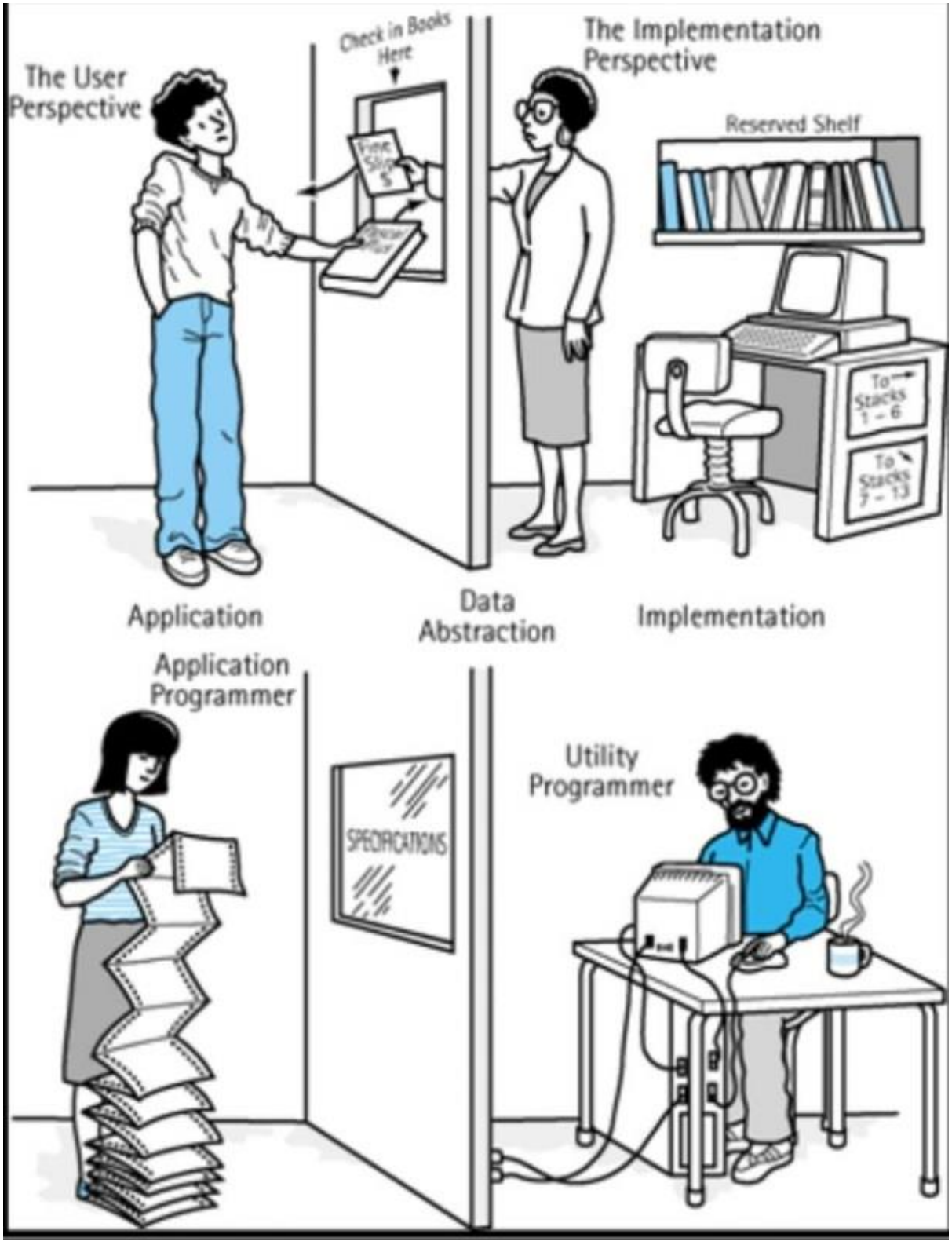
-getter method allows us to get or return the initialized value of speed

ABSTRACTION

An application of oop that is used to hide complexity of a system/code and only showing the main functionality to the user.

TAKE NOTE: whenever Abstraction is used it's indicated or is shown by the use of the key word abstract.

-An abstract method cannot have a body


The User Perspective
Check in Books Here
The Implementation Perspective
Reserved Shelf
To Stacks 1 - 6
To Stacks 7 - 13
Application
Data Abstraction
Implementation
Application Programmer
SPECIFICATIONS
Utility Programmer

CODE SNIPET:

```
abstract class Animal {
    // Abstract method
    public abstract void sound();
}

class Dog extends Animal {
    public void sound() {
        System.out.println("Barks");
    }
}
```

The code above shows the work of abstraction by hiding implementation of the sound() method in the abstract class but its complexity has been implemented on the Dog class

INHERITANCE

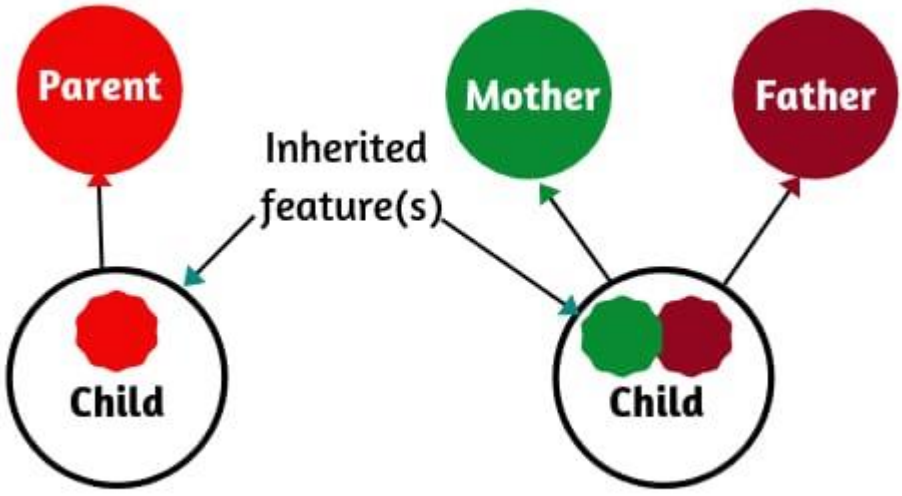


Fig: Inheritance of features in the real world

DEFINITION: an oop principle that allows a subclass (child class) to acquire properties and behavior of a parent class.

-when you see the keyword **extends** just take not that a subclass has been created and can acquire properties of the parent class

-it also supports polymorphism where the child class can override methods of the parent class

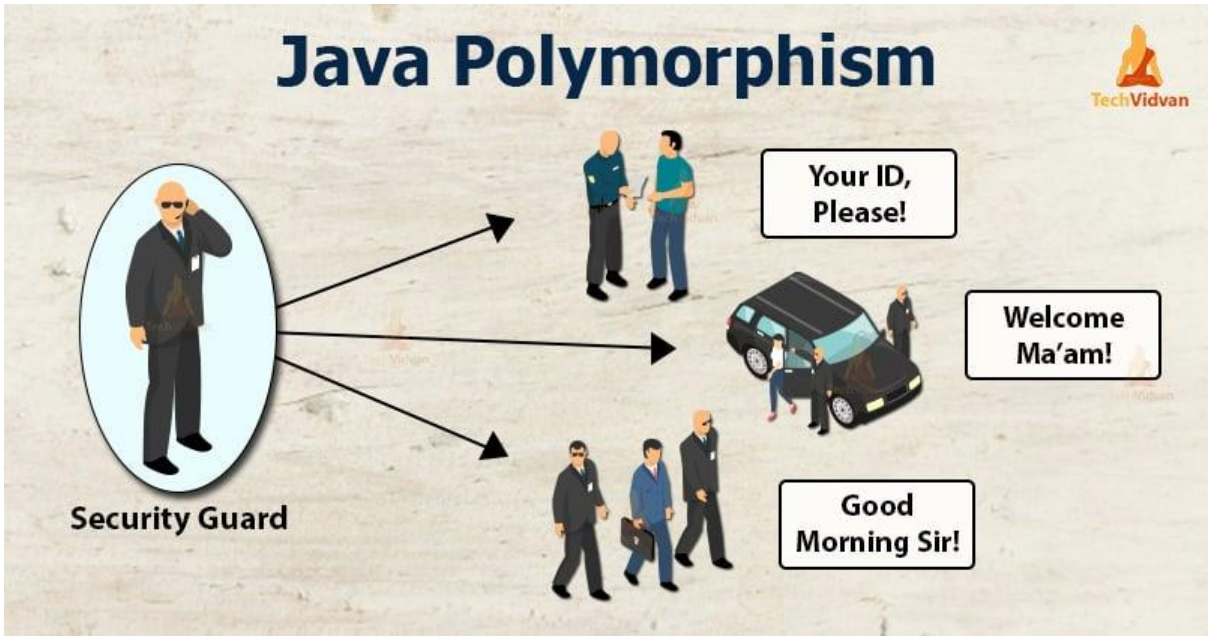
CODE SNIPET:

```
class Vehicle {
    public void move() {
        System.out.println("The vehicle is moving");
    }
}

class Car extends Vehicle {
    // Car can use Vehicle's move method and can also have its own methods
    public void fuelType() {
        System.out.println("The car uses petrol");
    }
}
```

Inheritance is a concept in programming that allows one class (like Car) to take on the properties and behaviors of another class (like Vehicle). Think of it like a child inheriting traits from their parent. In the same way, a class can inherit features from another class, so you don't have to repeat the same code every time.

POLYMORPHISM



MR JOHN is a security guard who has three forms 1st his a security card at a certain shop not only that but his a security guard for Mr. and Mrs. Smith

DEFINITION: The ability of an object to take different underlying forms (data types)

There are two types of polymorphism:

1. COMPILE TIME POLYMORPHISIM (Method Overloading)
2. RUN TIME POLYMORPHISIM (Method Overriding)

1. Method Overloading

What is it?

Method Overloading is when multiple methods in the same class have the same name but different parameters (either in number, type, or both). The methods perform similar but slightly different tasks, depending on what arguments are passed.

This is like having a person named John (Security guard) who knows how to do different things based on what you ask him:

If you give John one object, he knows how to work with it in one way.

If you give him two objects, he knows how to work with them in a different way.

Key Points:

Same method name, but different parameters.

Can change the number of parameters, type of parameters, or both.

It is a form of compile-time polymorphism (also called static polymorphism) because the decision of which method to call happens at the time of compiling the code.

Example of Method Overloading:

```
class Calculator {
    // First method: adds two integers
    public int add(int a, int b) {
        return a + b;
    }

    // Overloaded method: adds three integers
    public int add(int a, int b, int c) {
        return a + b + c;
    }

    // Overloaded method: adds two doubles
    public double add(double a, double b) {
        return a + b;    } }
```

2. Method Overriding

What is it?

Method Overriding is when a method in a child class (subclass) has the same name, return type, and parameters as a method in its parent class (superclass). However, the implementation (body) of the method is different in the child class.

This allows the child class to provide its own specific implementation of a method that is already defined in its parent class.

Key Points:

Same method name, same parameters, but different implementation.

Happens between a parent class and a child class.

It is a form of run-time polymorphism (also called dynamic polymorphism) because the decision of which method to call happens during program execution, depending on the object type.

Example of Method Overriding:

```
class Animal {
    // A method in the parent class
    public void sound() {
        System.out.println("The animal makes a sound");
    }
}
class Dog extends Animal {
    // Overriding the parent class method
    @Override
    public void sound() {
        System.out.println("The dog barks");  } }
```

Take note that Sound () method was changed from its predefined function to a new form.

Differences between Method Overloading and Method Overriding

Real-Life Analogy

Method Overloading: Think of someone with the same name performing different tasks based on what you give them. Imagine asking a cashier named Jane. If you give Jane cash, she processes it one way; if you give her a card, she processes it another way. Jane's name (the method name) is the same, but she works differently based on what you give her (the parameters).

Method Overriding: Imagine a restaurant chain where every branch has the same menu (methods from the parent class), but each branch makes the dishes (methods) differently. For example, if you visit a Mexican restaurant or an Italian restaurant from the same chain, they may serve food differently, even if the dish name is the same.

THE END!!