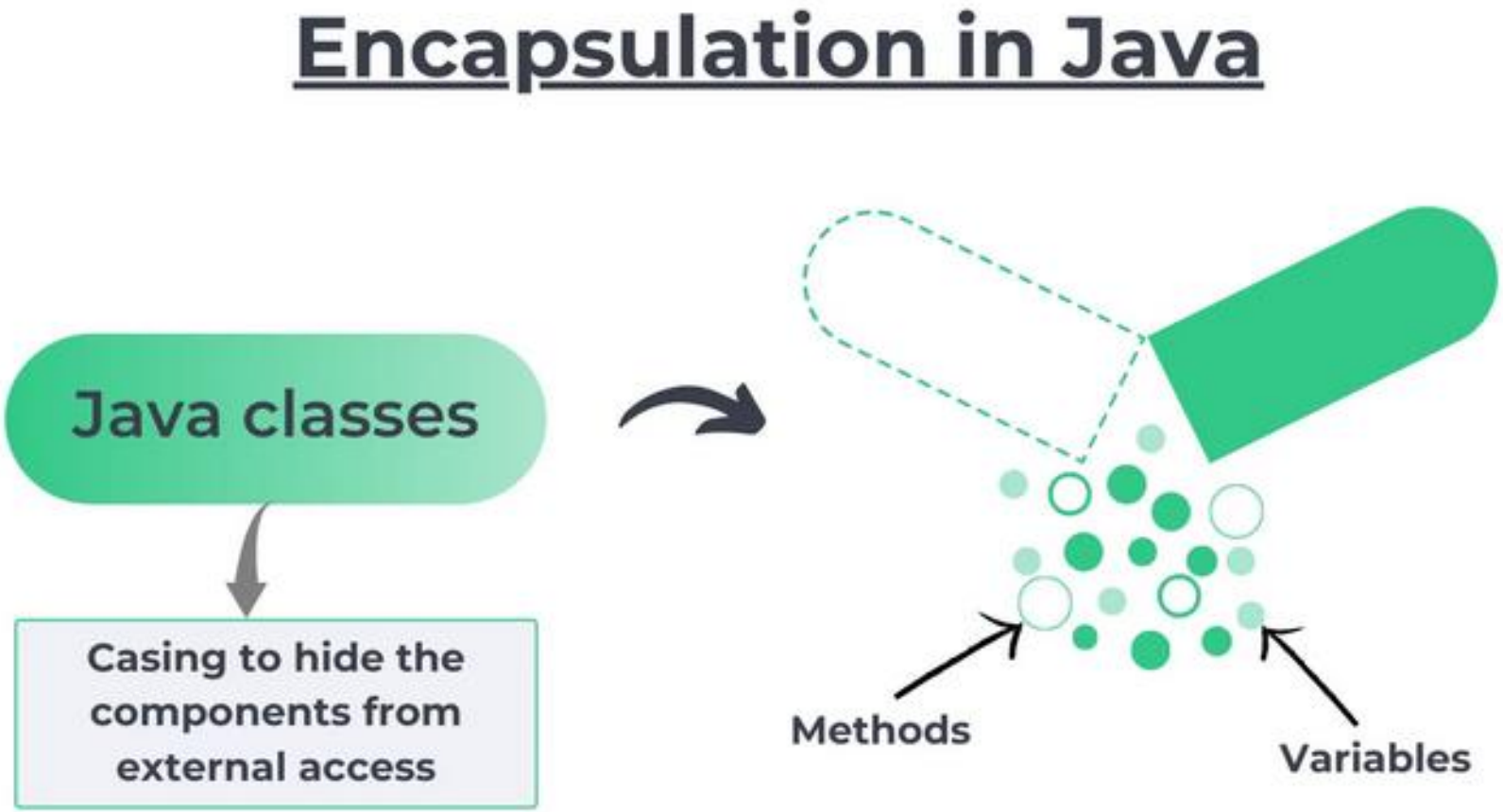
Encapsulation



Encapsulation

- **Encapsulation** is one of the four fundamental OOP concepts. The other three are inheritance, polymorphism, and abstraction.
- Encapsulation in Java is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as **data hiding**.
- To achieve encapsulation in Java −
- Declare the variables of a class as private.
- Provide public setter and getter methods to modify and view the variables values.

Example

The public setXXX() and getXXX() methods are the access points of the instance variables of the EncapTest class. Normally, these methods are referred as getters and setters. Therefore, any class that wants to access the variables should access them through these getters and setters.

```
/* File name : EncapTest.java */
public class EncapTest {
   private String name;
   private String idNum;
   private int age;

   public int getAge() {
      return age;
   }
   public String getName() {
      return name;
   }
   public String getIdNum() {
      return idNum;
   }
   public void setAge( int newAge) {
      age = newAge;
   }
   public void setName(String newName) {
      name = newName;
   }
   public void setIdNum( String newId) {
      idNum = newId;
   }
}
```

The variables of the EncapTest class can be accessed using the following program −

```
/* File name : RunEncap.java */
public class RunEncap {

   public static void main(String args[]) {
      EncapTest encap = new EncapTest();
      encap.setName("James");
      encap.setAge(20);
      encap.setIdNum("12343ms");

      System.out.print("Name : " + encap.getName() + " Age : " + encap.getAge());
   }
}
```

Output

Name : James Age : 20

Benefits of Encapsulation

- The fields of a class can be made read-only or write-only.
- A class can have total control over what is stored in its fields.

Encapsulation characteristics

- It is a way of combining various data members and member functions that operate on those data members into a single unit

Explanation: It is a way of combining both data members and member functions, which operate on those data members, into a single unit. We call it a class in OOP generally. This feature have helped us modify the structures used in C language to be upgraded into class in Java languages.

- If data members are private, what can we do to access them from the class object?
- Create public member functions to access those data members

Explanation: We can define public member functions to access those private data members and get their value for use or alteration. They can't be accessed directly but is possible to be access using member functions. This is done to ensure that the private data doesn't get modified accidentally.

- While using encapsulation, Data member’s data type can be changed without changing any other code

Explanation: Data member’s data type can be changed without changing any further code. All the members using that data can continue in the same way without any modification. Member functions can never change the data type of same class data members.

Immutable Class

Encapsulation helps in writing immutable classes in java.

An object is immutable when its state doesn't change after it has been initialized. For example, String is an immutable class and, once instantiated, the value of a String object never changes.

Creating an Immutable Class in Java

To create an immutable class in Java, you need to follow these general principles:

- Declare the class as final so it can't be extended.
- Make all of the fields private so that direct access is not allowed.
- Don't provide setter methods for variables.
- Make all mutable fields final so that a field's value can be assigned only once.
- Initialize all fields using a constructor method performing deep copy.
- Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

Example of Immutable class

```
class Main {
  public static void main(String[] args) {

    // create object of Immutable
    Immutable obj = new Immutable("Programiz", 2011);

    System.out.println("Name: " + obj.getName());
    System.out.println("Date: " + obj.getDate());
  }
}
```

Output

```
Name: Programiz
Date: 2011
```

**We will learn more about this in the next appropriate class

```
// class is declared final
final class Immutable {
  // private class members
  private String name;
  private int date;
  Immutable(String name, int date) {
    // class members are initialized using constructor
    this.name = name;
    this.date = date;
  }
  // getter method returns the copy of class members
  public String getName() {
    return name;
  }
  public int getDate() {
    return date;
  }
}
```

The data which is prone to change is near future

Explanation: The data prone to change in near future is usually encapsulated so that it doesn't get changed accidentally. We encapsulate the data to hide the critical working of program from outside world.

Encapsulation be achieved using Access Specifiers

Explanation: Using access specifiers we can achieve encapsulation. Using this we can in turn implement data abstraction. It's not necessary that we only use private access.

> *"Data abstraction is the process of hiding certain details and showing only essential information to the user. Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next classes)."*

- Global variables violates the principle of encapsulation almost always

Explanation: Global variables almost always violates the principles of encapsulation. Encapsulation says the data should be accessed only by required set of elements. But global variable is accessible everywhere, also it is most prone to changes. It doesn't hide the internal working of program.

Using access declaration for private members of base class would destroy the encapsulation mechanism if it was allowed in programming

Explanation: If using access declaration for private members of base class was allowed in programming, it would have destroyed whole concept of encapsulation. As if it was possible, any class which gets inherited privately, would have been able to inherit the private members of base class, and hence could access each and every member of base class.

This code violates encapsulation

Explanation: This code violates the encapsulation. By this code we can get the address of the private member of the class, hence we can change the value of private member, which is against the rules.

```
class student
{
    int marks;
    public : int* fun()
    {
        return &marks;
    }
};
main()
{
   student s;
   int *ptr=c.fun();
   return 0;
}
```

Encapsulation is the way to add functions in a user defined structure.

Explanation: False, because we can't call these structures if member functions are involved, it must be called class. Also, it is not just about adding functions, it's about binding data and functions together.