In Java, dynamic polymorphism, also known as runtime polymorphism, is achieved through method overriding. It allows a subclass to provide a specific implementation of a method that is already defined in its superclass. This enables the Java runtime to determine which method to execute based on the actual object type, rather than the reference type.

However, dynamic polymorphism does not apply to instance variables or data members. Here's why:

Instance Variables and Polymorphism
Static Binding: Instance variables are bound at compile time, not at runtime. This means that the reference type determines which variable is accessed, not the actual object type.

No Overriding: Unlike methods, instance variables cannot be overridden. If a subclass declares a variable with the same name as a variable in its superclass, it shadows the superclass variable rather than overriding it.

Example to Illustrate
Let's look at an example to demonstrate this behavior:

Java

Collapse

```
class SuperClass {
    int value = 10; // Instance variable

    void display() {
        System.out.println("SuperClass value: " + value);
    }
}

class SubClass extends SuperClass {
    int value = 20; // Shadows the superclass variable

    @Override
    void display() {
        System.out.println("SubClass value: " + value);
    }
}

public class PolymorphismExample {
    public static void main(String[] args) {
        SuperClass superClass = new SuperClass();
        SubClass subClass = new SubClass();
        SuperClass ref = new SubClass();

        System.out.println("superClass.value: " + superClass.value); // Output: 10
        System.out.println("subClass.value: " + subClass.value);     // Output: 20
        System.out.println("ref.value: " + ref.value);               // Output: 10

        superClass.display(); // Output: SuperClass value: 10
        subClass.display();   // Output: SubClass value: 20
        ref.display();        // Output: SubClass value: 20
    }
}
```

Explanation
Method Overriding: The display method is overridden in SubClass. When display is called on ref, which is a SuperClass reference pointing to a SubClass object, the SubClass version of display is executed. This is dynamic polymorphism.

Instance Variable Shadowing: The value variable in SubClass shadows the value variable in SuperClass. When accessing value through a reference, the reference type determines which value is accessed. Thus, ref.value accesses SuperClass.value, not SubClass.value.

Conclusion
Dynamic Polymorphism: Achieved through method overriding, allowing the runtime to determine which method to execute based on the actual object type.

Instance Variables: Do not participate in dynamic polymorphism. They are accessed based on the reference type, leading to static binding.