**OOPS:**

**Data Hiding:**

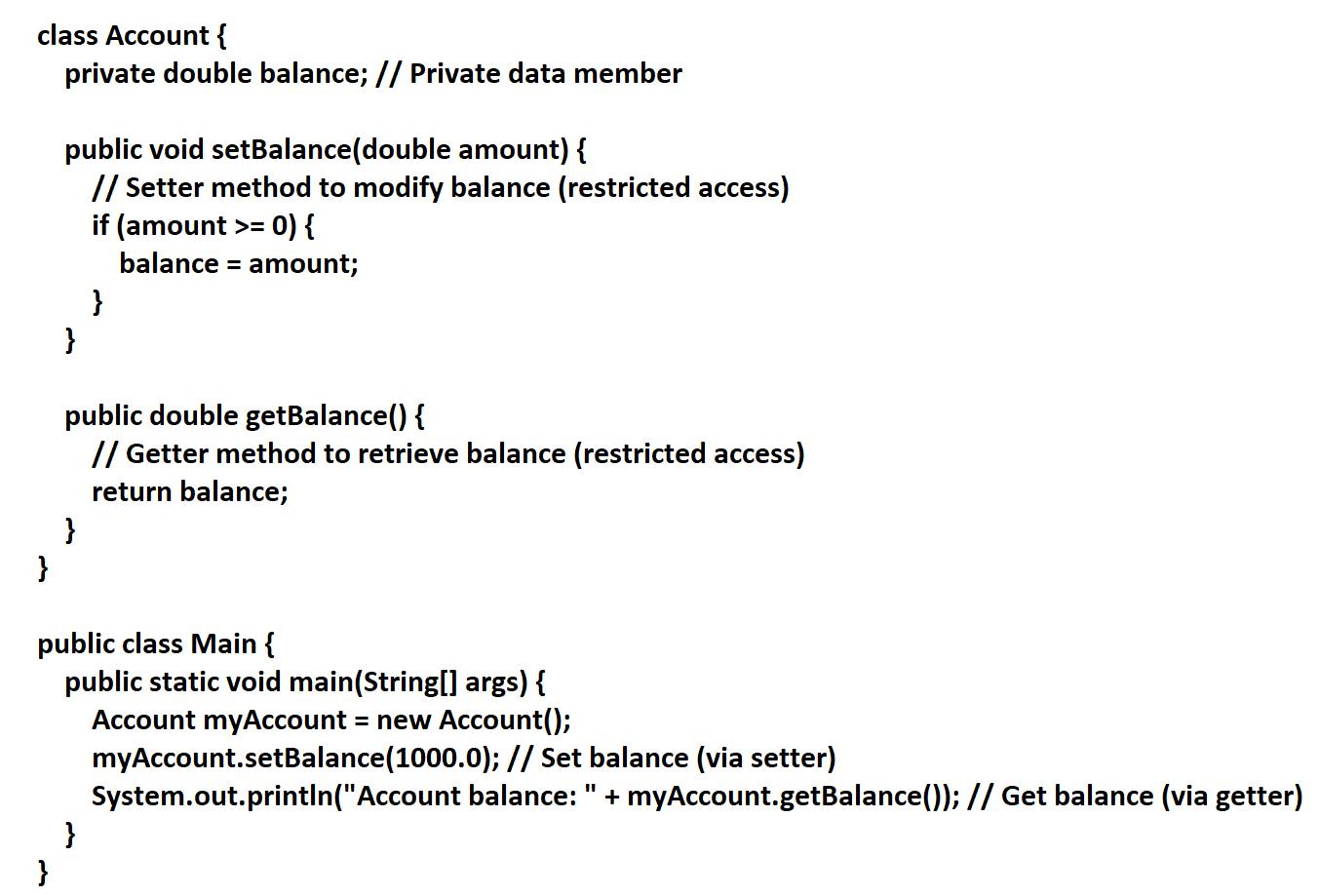
Data hiding in Java is a fundamental concept in object-oriented programming.

Data hiding involves encapsulating data within a class and restricting access to other entities using access modifiers (like private modifier).

It ensures that the internal details such as data members of an object are hidden from external entities.

By controlling access to class members, data hiding maintains data integrity and code security.

Data hiding is essential for modular design, preventing unintended interference from external entities, and maintaining a clean separation between interface and implementation details in your Java code



In this example, balance is hidden from direct access outside the Account class, ensuring data security.

**Abstraction:**

**Abstraction** in Java is a fundamental concept in OOP programming that allows developers to hide implementation details while exposing only set of services to users.

It focuses on what an object does rather than how it accomplishes it.

In Java, abstraction is achieved using abstract classes and interfaces.

By Abstraction we achieve:

* **Security**: Hide implementation details and expose only essential information.
* **Modularity**: Separate interface from implementation.
* **Flexibility**: Allow different implementations for the same interface.

**Encapsulation:**

**Encapsulation** allows developers to **bundle code(methods) and data(variables) together** into a single unit, providing a clean interface for interacting with objects. It restricts direct access to the internal details of an object from outside the class.

Encapsulation = Databinding + Abstraction

**Advantages:**

**Data Hiding**: By making data members private, we prevent direct access to them from other classes. Only methods within the class can modify or retrieve the data.

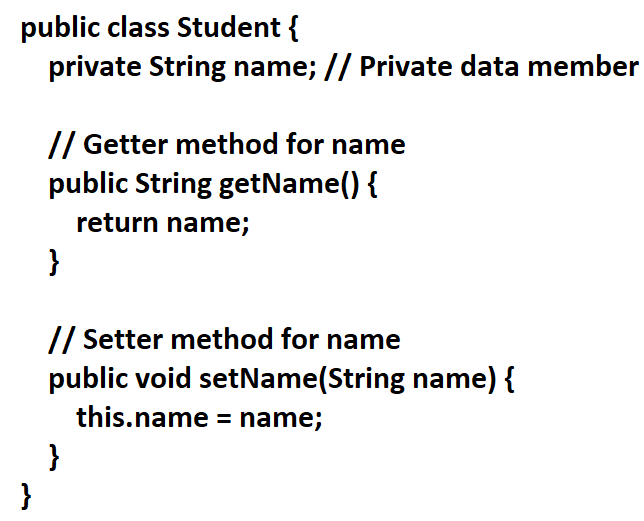
**Control Over Data**: We can enforce rules and logic **within setter methods**. For example, ensuring that an id value is greater than 100 or disallowing negative numbers.

**Security**: Private data members are inaccessible from external classes, enhancing data security.

**Unit Testing**: Encapsulated classes are easier to test in isolation.

**Readability and Maintainability**: Clear separation of interface and implementation simplifies code maintenance.

**Flexibility:** It provides flexibility to the user to use system very easily.



In this example:

* The name data member is private, ensuring data hiding.
* The getName() method retrieves the name (**read-only access**).
* The setName(String name) method sets the name (**write-only access**).

What is a tightly encapsulated class?

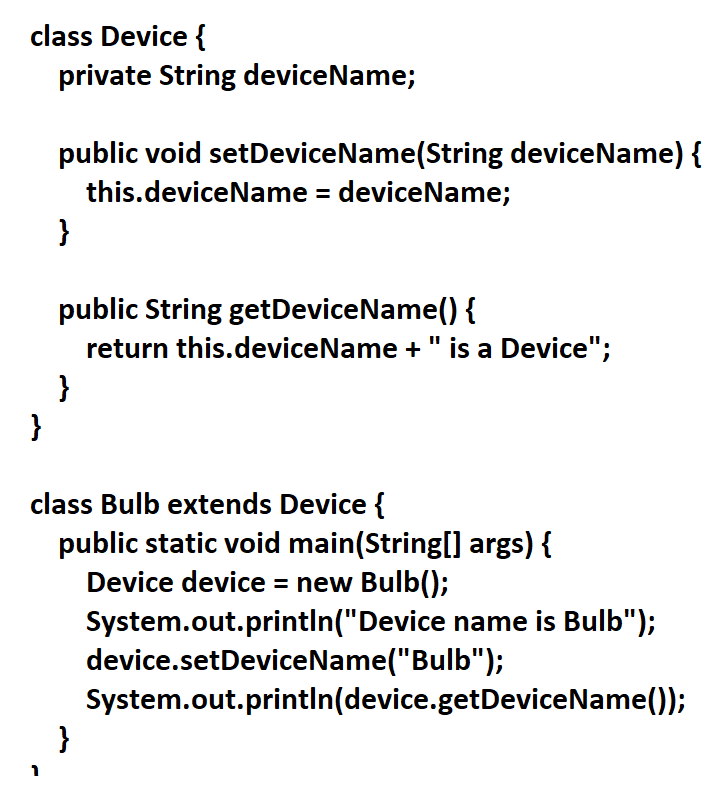
A class is said to be tightly encapsulated if and only if every variable of that class declared as private whether the variable has getter and setter methods are not.

**IS-A relationship:**

* Also known as **inheritance relationship.**
* **When one class inherits another class, it is called an IS-A relationship.**
* The IS-A relationship is achieved using the **extends** keyword**.**
* It represents a type of relationship between two classes.
* It implies that a subclass is a type of its superclass.
* The superclass provides common functionality, and subclasses extends it.

Ex: ‘Dog’ is subclass of ‘Animal’

Advantages: **reusability** and **avoids redundancy**



In this example, the Bulb class inherits the Device class. Hence, we can confidently say that “**Bulb IS-A Device.**”