Week 9 – Webinar B (Access Modifiers)

In this webinar session, we will look at how we can hide the internal elements (e.g. attributes and methods) of a class by using access modifiers.

# Learning Objectives

* Understand the principles of encapsulation
* Enforce data hiding through the use of access modifiers

# Access Modifiers in Java

Since we have gone over to IntelliJ, we have seen the public keyword in a couple of places: (1) as part of the class declaration (e.g. **public** class Weapon) and (2) the main method declaration (**public** static void main(String[] args).

**public** is just one of several access modifiers we have access to in Java. Table 1 provides a quick overview of the access modifiers in Java, including the “visibility level” associated with each. For example, if a method is declared **private**, that method can **only** be used within the class itself.

We’ll mainly focus on **public** and **private** in this webinar, as they are the two different extremes of access control that we can provide.

Table 1 Member-level access modifiers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Modifier (as applied to an attribute or method)** | **Class** | **Package** | **Subclass** | **World** |
| public | ✓ | ✓ | ✓ | ✓ |
| protected | ✓ | ✓ | ✓ | ✗ |
| none (also known as **default** or **package-private**) | ✓ | ✓ | ✗ | ✗ |
| private | ✓ | ✗ | ✗ | ✗ |

# Step 1 – Creating a Basic BankAccount Class

We’ll create a BankAccount class with the attributes and methods depicted in the UML Class Diagram in Figure 1.

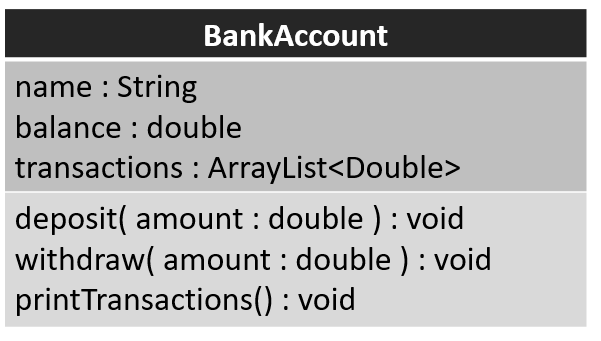


Figure BankAccount UML Class Diagram

# Step 2 – BankAccount Object

Once we have the class defined, we will create an object of the BankAccount class to test that the methods work.

We’ll see that although the methods do work, because the instance variables (**name**, **balance**, and **transactions**) are all public, anyone can “go into” these variables for any object and start messing with them as they please – completely bypassing the methods we’ve defined.

# Step 3 – Implementing Access Control

Instance variables should always be private – we don’t want anyone to have the ability to reach into an object and begin changing its state – they must interact with the object through its public methods.

Variables that are marked private can only be accessed within the class they are declared (i.e. code from outside the class won’t be able to access them). Make the three BankAccount instance variables **private** and observe what happens to any code in your main method that attempts to use any of these variables in any way.

Figure 2 illustrates how access modifiers can be represented in UML, where public members/attributes are prefixed with a +, and private members/attributes are prefixed with a –.

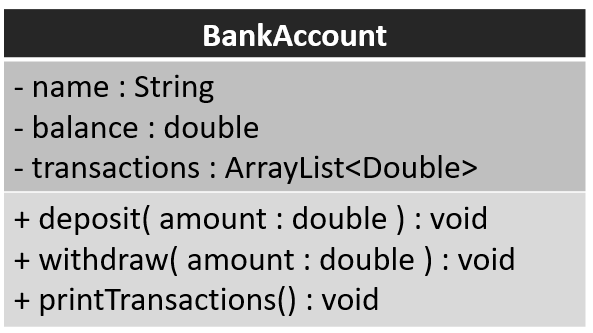


Figure BankAccount UML Class Diagram (with access modifiers)

We can have private methods too. It doesn’t make much sense in this example since we want to be able to make deposits, withdrawals, and print the transactions. But if you look back at code you’ve written previously, are there any methods that are only used within the class? In your assessmen, for example, you most likely have an update method in a class that calls move and render – if the update method is the only method that is used outside of your class, and you don’t want someone to be able to call render or move, you could mark those private and allow access to just the update method (marked public).