Java How to Program

Early Objects

TENTH EDITION

Paul Deitel & Harvey Deitel

Chapter 3 Introduction to Classes, Objects, Method and String

Introduction

We looked previously about how to take object from predefined class that java has and from this object we use the methods associated with this object, but what we doing here is to create our own class and use some objects from this class, then we can use the methods we will create inside.

We will use simple real world example we will create from scratch which is Class **Account**, as we know each account is associated with some private data like the balance and the account holder.

We will use for now instance variable for the account holder and its a type string, the **instance variable** maintain data for each object on its own.

// Account.java // Account class that contain a name instance vraible // and methods to set and get its value public class Account { private String name; // name: instance private variables /* setName: Set the account username. */ public void setName(String name) { this.name = name; } /* getName: Return the account username. */ public String getName() { return this.name; } } // End of Account class

Class Declaration

Public class Account{

} // End of Account class

Every Java program should contain at least one class

- public, is access modifier we will go in deeper in this but not now
- class, is the keyword to start your declartion of a class
- Account, is the class name and start with capital letter and the file name should be as same as class name with .java as extenation.
- The braces is the start and end of the class body

  So java class is start with class keyword followed by the name of the class which start with capital letter and should you have the file with same name followed by .java

Java is case sensitive so Welcome1 is a class name and welcome1 is another class.

Instance Variable

Each object(instance of class) has its own instance variable which maintain the object data, these instance variable are declared **inside class** but **outside methods**, and more convenient way to put it before methods.

Private String name, private is another access modifier like public, but we should know for now that any private instance variables are only accessible to method of the class, so it can not directly accessed by the object you take from the class its accessed by the methods of the object.

setName Methods of class Account

public void setName(String name) **==> This line refer to Method header** {

this.name = name;

}

The method header contain:

Returned Type, which before the name of method **setName**, this type till the method which type of information should return to its caller, but here the keyword **void** means this method return no information to its caller and as we can see by the end of method there is no **return keyword**, it just do some operation.

Method Paramters, the **setName** receive an parameters when the method call and we can see these parmater between the **parenthes**, if there are multiple parameter **parameter list**, we should seprate each of them using **,** comaa, then each of these paramter should contain the **type** in this case **string** and the name of the variable in this case **name**.

After the **Method Header**:

We have the method body closed in {}, **braces**, which may contain multiple statements but as we can see here it just contain one statement.

Note !

The **name** parameter is local variable while the **name** instance variable are global variable where you can use inside anywhere in the class, but if the method have two variable with same name, one local and other is instance of the class the method here use its local, so we refer to the instance by the word **this** which associated with the object and use the instance variable of the class.

getName Methods of class Account

public String getName() // Method header { return this.name; }

We can notice here the return type is **String** then this method should contain in the end the return keyword, and should return a **string**, and as we can see it return the name instance variable which is String.

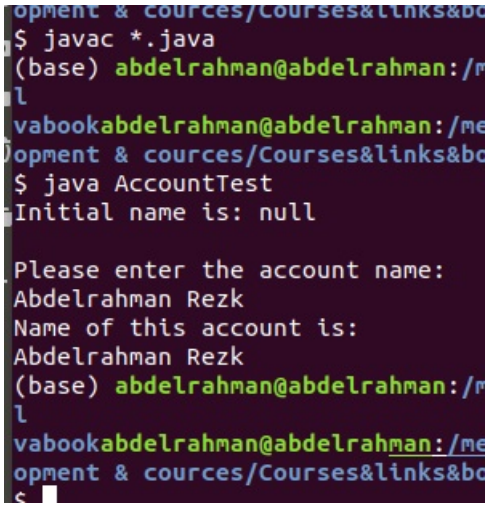
The **return** keyword pass the **string name** to the methods caller.

Account Test

We can not use the Account class hence we do not have the main method yet, so in most case we use a **Driver Class** to test the class we create to separate your work and this Test class will contain the main method, or we will need to create the main method inside the **Account Class**

// AcountTest.java // Creating and mainpulting an Account object import java.util.Scanner; public class AcountTest { // create a scanner object Scanner input = new Scanner(system.in); //Create account object Account myAccount = new Account(); // Display intial value of the name System.out.printf("Initial name is: %s%n%n", myAccount.getName()); //Prompt for read System.out.println("Please enter the account name:"); String name = input.nextLine(); myAccount.setName(name); // set the account name System.out.printf("Name of this account is: %n%s%n", myAccount.getName()); }

javac *.java for compile multiple files



default types

Each type of data has its own default value, and for string as we can see when we call the method get name we get null because there is no name for this object yet, but once we set this instance variable and call the method again, we can see the result.

The **Scanner Object**, as we talked before scanner object have different methods for different data types, we work with integer before with method **nextInt** but here we have **nextLine**, which accept and characters you write until you press the **enter character**, then this method return to its caller the text you write including anything except the **enter charater**.

How calls work

When you call some any method in the main method this is actually happen:

- The app transfer execution from which line in the main, to the method that you call.
- After this method perform its task, the caller wait to finish then if the method return it will take the value to the caller, otherwise wait the method to finish, then return to next line after the call.

UML Graph

<<>> Define constructor in UML

- **-** Define Private status
- **+** Define public status

Account class that contain: Everything created is defined at the beginning of the function or beside the instance variables.

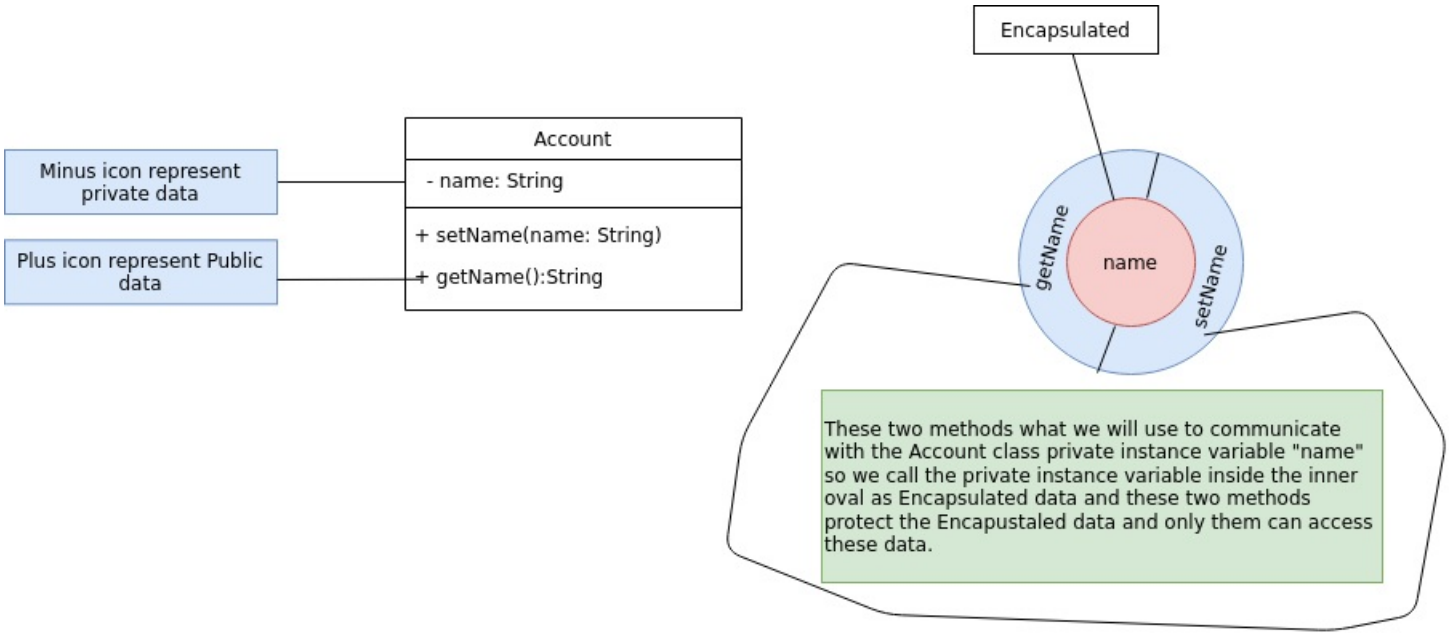
Instance private Variables:

```
- name: String
```

Constructors: public Methods:

```
+ setName(name :String)
+ getName(): String
```

Graph represent UML associated with conceptual view of an Account objects with Encapsulating Data



Primitive Types Vs. Reference Type

Primitive lost the values its hold when you assign new values which means it can only hold one value at a time, and these Primitive type are initialized by default value which is 0 for most of them, like int, double, float, char and for boolean is false, but we should know that these default values are assigned when you create the instance variable without assign initialization value, and also **Local Variable are not initialized by default value**

Reference Type like **String** classes you create, or object you use like **Scanner** are all of Reference Types.

Account Class: Initializing objects with constructors

Each class you declare can optionalyy provide a constructor with paramters that can be used to inilize an object of the class when the object is created, and actually when you do not create any constructor java create a deafult one for you which set the instance variables of your object to its default values as we saw above **null** for String.

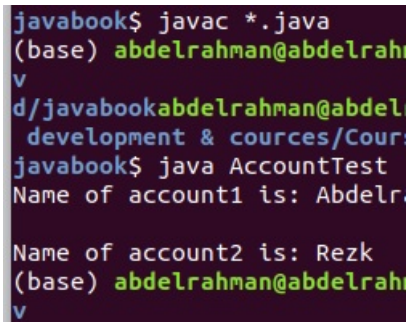
// Account.java // Account class with constructor that initializes the name. public class Account { private String name; // name: instance private variables // constructor initializes name with paramter name public Account(String name) { this.name = name; } /* setName: Set the account username. */ public void setName(String name) { this.name = name; } /* getName: Return the account username. */ public String getName() { return this.name; } } // End of Account class

Account Constructor Declaration

To create constructor just you same as method but without **return** and should have the same name of the class, but is different in the parameters.

Then this paramter will be passed when you create an object from the Account class, then the constructor will set this name to **name instance variable**.

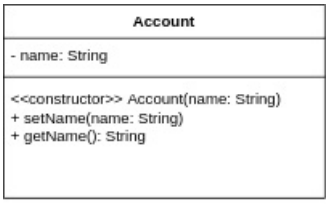
// AcountTest.java // Creating and mainpulting an Account object import java.util.Scanner; public class AccountTest { public static void main(String[] args) { // create a scanner object Scanner input = new Scanner(System.in); //Create 2 account object Account account1 = new Account("Abdelrahman"); Account account2 = new Account("Rezk"); // Display intial value of the name System.out.printf("Name of account1 is: %s%n%n", account1.getName()); System.out.printf("Name of account1 is: %s%n", account2.getName()); } }



Note !

Once we are create any constructor in the class, then there is no default construcor java will provide you because of that you should respect the constructor when you take an object from.

UML with constructor

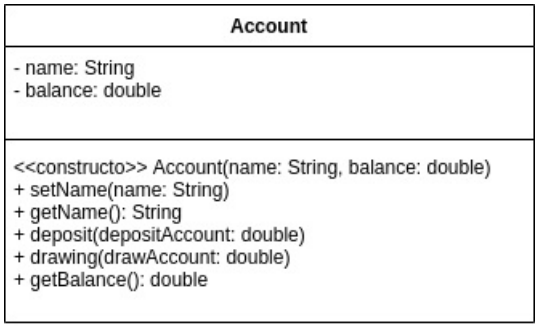


Account Class with Balance, Floating-point

After we have know more about how to create class, take object from the class, know about instance variables, methods and uml, now we can make the Account class more bigger by include the balance for each Account, this will require some method like check balance, withdraw from your account and others.

UML

Once we have the **UML** we should know how to create the class, uml is a big picture of what we need then we just convert this uml to code in no time.



Floating-point

double is another primitive data type which hold default value 0.0 and as we can see this data type can hold numbers with floating point like 10.8 and others nad its size is 8-byte.

<<>> Define constructor in UML

- **-** Define Private status
- **+** Define public status

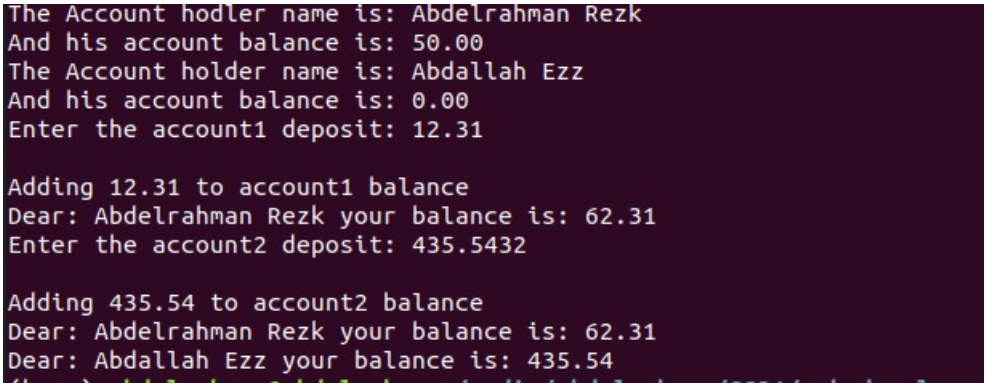
Account class that contain: Everything created is defined at the beginning of the function or beside the instance variables.

```
Instance private Variables:
    - name: String
    - balance: double
Constructors:
    <<Constructor 1>> Account(name: String)
public Methods:
    + setName(name :String)
    + getName(): String
    + deposit(depositAccount: double)
    + drawing(drawAccount: double)
    + getBalance(): double
The instance private variable name can be accessed throughout the method.
also, we may not use the keyword this and make name = name direct but it's not good practice
```

Note !

The code is documented with multiple of comments

public class Account{ // name: instance private variables private String name; private double balance; // Account Constructor 1 public Account(String name, double balance){ /* Constructor 1: That set name to default value that user pass when create an Object from Account Class. */ this.name = name; if(balance > 0) this.balance = balance; } /* ------------------------- Start methods that access private variable name */ public void setName(String name){ /* setName: Set the account username. */ this.name = name; } public String getName(){ /* getName: Return the account username. */ return name; } /* ------------------------- End methods that access private variable name ------------------- */ /* ------------------------- Start methods that access private variable balance ------------------- */ public void deposit(double depositAccount){ /* deposit: Add new ammount of balance to the current ammount of user account. */ if(depositAccount > 0) // Valid deposit will be add other not because of negative values balance += depositAccount; } public void drawing(double drawAccount){ /* drawing: Decrease the current balance of user money.. */ if(drawAccount <= this.balance) // Valid if user has the amount of money need to draw this.balance -= drawAccount; } public double getBalance(){ /* getBalance: Return the current user balance. */ return this.balance; } /* ------------------------- End methods that access private variable balance ------------------- */ } import java.util.Scanner; public class AccountTest{ /* Account Test that creating and maintain the Account class */ public static void main(String [] args){ Account account1 = new Account("Abdelrahman Rezk", 50.00); Account account2 = new Account("Abdallah Ezz", -50.00); System.out.printf("The Account hodler name is: %s%nAnd his account balance is: %.2f%n", account1.getName(), account1.getBalance()); System.out.printf("The Account holder name is: %s%nAnd his account balance is: %.2f%n", account2.getName(), account2.getBalance()); // Create scnner object Scanner input = new Scanner(System.in); // Read from user keyboard System.out.print("Enter the account1 deposit: "); double depositBalance = input.nextDouble(); System.out.printf("%nAdding %.2f to account1 balance %n", depositBalance); account1.deposit(depositBalance); System.out.printf("Dear: %s your balance is: %.2f%n", account1.getName(), account1.getBalance()); System.out.print("Enter the account2 deposit: "); depositBalance = input.nextDouble(); System.out.printf("%nAdding %.2f to account2 balance \n", depositBalance); account2.deposit(depositBalance); System.out.printf("Dear: %s your balance is: %.2f%n", account1.getName(), account1.getBalance()); System.out.printf("Dear: %s your balance is: %.2f%n", account2.getName(), account2.getBalance()); } }



GUI for Display

Instead of using terminal to represent result we use GUI to represent your message in. Java provide swing library with its JOptionPane class to help us represent this GUI. Dialog box is just like the Gmail you use to write your e-mail and send to some one, also its looks like the window of Firefox and other programs.

// Dialog1.java // Using JOptionPane to display Multiple lines in a dialog box. import javax.swing.JOptionPane; public class Dialog1 { public static void main(String [] args) { // Display a dialog with message JOptionPane.showMessageDialog(null, "Welcome To Java GUI"); } // End of Main method } // End of Dialog Class

JOptionPane class static method showMessageDialog

The program use JOptionPane class from package javax.swing, this package contain different classes for helping you create GUI, some GUI acting to take input from user in some inputs, but here showMessageDialog method is static method that disply to user a message also, in first paramters which was **null** we can define the place on the screen to display the message.

static Method is provided by designer of class JOptionPane to help you to perform task without **reinventing the wheel**, just call the method, and as we can see static method is called without take an object from the class, and this because it's available to any class to use this method not just JOptionPane.

GUI with user entry

Also the JOptionPane provide you with static method that help you to enter some data.

// NameDialog.java // Obtaining user input from a dialog import javax.swing.JOptionPane; public class NameDialog { public static void main(String [] args) { // Prompt the message String name = JOptionPane.showInputDialog("What is Your Name ??"); // Create the message String message = String.format("Welcome, %s to Java GUI !!!!!", name); // diesplay the name JOptionPane.showMessageDialog(null, message); } // End of Main method } // End of class NameDialog

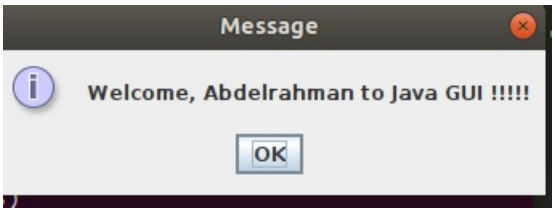
showInputDialog

Help user to enter some data and we can retrieve this data because the method is return the data that use input and we receive it in string then using **String format** we have design the message to display it using the **showMessageDialog** method.

String.format is same as **system.out.printf**.

- Input Name

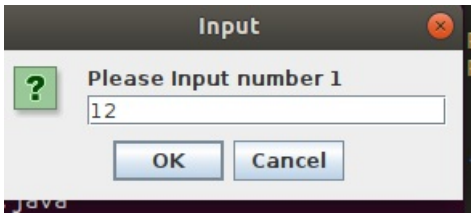
- Display Result



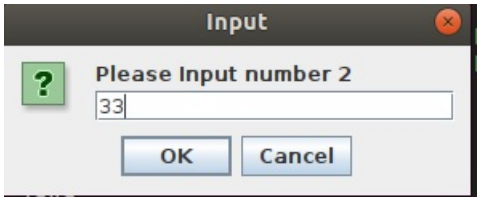
Another App using GUI for Addition // AdditionGUI.java // AdditionGUI program that inputs two numbers and displays their sum. import javax.swing.JOptionPane; public class AdditionGUI { // Main method ebgins execution of the program public static void main(String[] args) { // Prompt for input numbers int number1 = Integer.parseInt(JOptionPane.showInputDialog("Please Input number 1")); // first number to add int number2 = Integer.parseInt(JOptionPane.showInputDialog("Please Input number 2 ")); // second number to add int sum = number1 + number2; // sum of number1 and number2 // Format the string for displaying String message = String.format("The sum of %d and %d is: %d", number1, number2, sum); // The Dialog Message JOptionPane.showMessageDialog(null, message); } // End of main method } // End of class AdditionGUI

Method **parse Int** from Integer class is used to convert string number to integer since there is no other character except numbers in the string.

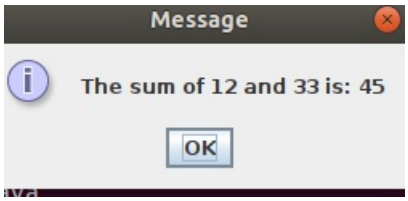
- Input number1



- Input number2



- Display their Sum



----------تم بحمد الله----------