Java Programming for Testing Automation Lab Guide

27697-54919-01

Java Programming For Testing Automation

Lab Guide

Written By: Rony Keren

Internet Team

John Bryce training center

Table of contents

© All rights reserved to John Bryce Training LTD

1. Exercise for modules 1 –5

1.1. Module 1: Getting started

Exercise 1

- Examine the JDK installation on your PC
- Verify that the JRE is also installed
- Check environment variables setting
- Open the command prompt and try to run java.exe & javac.exe
- Create a file named Hello.java
- Copy the following code and save the file:

```
public class Hello {

        public static void main (String [ ] args) {

                System.out.println (“ Hello Java World ! “ );

        }
}
```

- Compile the code via javac.exe in the command prompt
- Verify that the Hello.class file was generated
- Execute the program via java.exe

© All rights reserved to John Bryce Training LTD

1.2. Module 2: Packaging

Exercise 2

- Add to the Hello.java file a package declaration (at the top, before the class declaration):
  package app.core;
- Compile the source into a package (using javac –d)
- Run the program via specifying the fully qualified class name (app.core.Hello)

1.3. Module 3-4: Expressions & Flow control

All of the following exercises in this part should be implemented within main() method. No further classes or methods are required

Exercise 3 – output

1. Create a class named Printer
2. Define the following variables & initialize each with the specified values:
   - part1 – "There will be"
   - visitors – 5
   - part2 – "people for dinner."
3. Print the complete message
4. run the program an test the result
5. try to increment the number of visitors to 7 [visitors+2] in the print line
   - What happens when adding just visitors**+2** ?
   - What is the right way of updating the print line ?

© All rights reserved to John Bryce Training LTD

Exercise 4 – operators

1. Create a class that defines 2 random numbers and prints
    - each number
    - the sum of the numbers
    - the average value
    - the remainder when dividing each in 10
    - the area of a rectangle where one num is the width and the other is the height
    - In order to randomize values between 0-100 use the following: _int example = (int)(Math.random()*101);
    - Add a clear message to each printed value

Exercise 5 – conditions

1. Create a class that defines 2 random numbers and prints
    - the bigger value

2. Create a new class that defines a random number with a value between 0-100.
    - if the number is greater than 50 – print “Big !”
    - if the number is less then 50 – print “Small !”
    - if the number equals to 50 – print “Bingo !”

3. Create a new class that defines a random number with a value between 0-100.
    - if the value is between 0-50 – print “Small !”
    - else – print “Big !”

    in addition :
    - if the value is even (can be divided by 2) – print “Even”
    - else – print “Odd”

© All rights reserved to John Bryce Training LTD

4. Create a new class named "SalaryRaiser"

- define a random number named 'salary' with a value between 5000-6000.
- Now, raise the salary in 10% - only if the result is not greater than 6000 (which is the maximum salary allowed)
- print the current salary and the updated salary

5. Create a class that defines 3 random numbers and prints

- the bigger value

6. Salary taxes are calculated according to the following:

- 0- 23,000 nis -> tax rate is 10%
- 23,000- 50,000 nis -> tax rate is 20%
- 50,000- 100,000 nis -> tax rate is 30%
- 100,000 - up nis -> tax rate is 40%

Create a class named "TaxCalculator" that takes a salary of an employee (randomize a value to be used as an input) and prints the valid value after tax calculation

7. Create a class that randomize a value to present a year (like 970, 1990, 2010 …) and prints the year and if it is leap year or not.

- leap year must:
  - divide by 4
  - not divide by 100
  - if divided by 100 must also divide by 400

© All rights reserved to John Bryce Training LTD

Exercise 6 – loops

1. Create a class that defines a random number and prints all numbers from 1 to that number

2. Create a class that defines two random values and prints all values between them. note - which variable holds the higher value is not known.

3. Create a class that defines a random number and prints all even numbers from 0 to that number

4. Create a class that defines two random values ‘max’ and ‘den’ and prints all the numbers from 0 to ‘max’ that can be divided with ‘den’

5. Create a class that defines a random number with value between 0-10000 and print the following details with clear messages:
   - number of digits [4867 → 4]
   - the first left digit [6843 → 6]
   - sum of the number’s digits [473 → 14]
   - opposite order of the number’s digits [5892 → 2985]

6. Create a class that defines a random value between 0-100,000 and prints if it is a palindrome (a symmetric number like: 12321, 666, 47974, 404 …)

7. Create a class that defines a random number between 0-100 and prints the factorial value [4 → 1 X 2 X 3 X 4]

8. Fibonacci set is an array of numbers. Each number is the sum value of the two previous numbers. The first number is 1 [1,1,2,3,5,8,13,21,34,55,89…]

© All rights reserved to John Bryce Training LTD

Create a class that defines a random number named “index” with a value between 0-50 and prints the number in Fibonacci set that is located in the “index” position

9. Create a class that defines a random value between 0-50 and prints Fibonacci set from 1 to that value

10. Create a class named ‘Boom’ that implements the game “7-Boom” for all values from 0 to 100. The game rules are:
    - if the current number can be divided by 7 –print “boom”
    - if the current number has the digit ‘7’ – print “boom”
    - otherwise – print the number as is

1.4. Module 5: Java Arrays

Exercise 7

1. Create a class that creates an array[10] of numbers with random values between 0-100 and prints the total sum and the average

2. Create a class that creates an array[50] of numbers with random values between 0-100 and prints the highest value and its index in the array

3. Create a class that eliminates duplications. The class should be capable of getting an array with duplicated values and return an array of unique values generated from it. For example, for the input {1,2,5,1,6,1,5,4,8} the result should be {1,2,5,6,4,8}
    - create an array[10] of numbers with random values between 0-10
    - create an array with the required size to host the unique values
    - fill the unique array
    - print both arrays

© All rights reserved to John Bryce Training LTD

4. Create a class that reverse a given array order. For example, for the input {6,8,4,2,7,5} the result should be {5,7,2,4,8,6}.
    - create an array[10] of numbers with random values between 0-10 to be used as an input
    - print the array before and after reversing

5. Create a class that calculates student average year grade.
    - create a matrix according to the following:
        - there are 20 students in class
        - there are 10 different grades per student (randomize values between 0-100 as input)
    - print each student average grade
    - print the class average grade

© All rights reserved to John Bryce Training LTD

2. Exercise for modules 6 – 10

Bank System

System Description

Requirements:

The bank system holds a client list and allows them to deposit and withdraw cash money, manage accounts

Bank system provides the following:

- client list management
  - add/remove/update client
- account list management
  - clients can add/remove accounts
  - daily interest calculations – done automatically by the bank system
- Cash flow management
  - bank fortune – total amount taken from client deposits and accounts
  - Commission for each withdraw or deposit made by the clients
  - account balance daily update according to clients interest
- Activity log
  - all the following, that effects bank balance, are logged:
    - add / remove client
    - update client balance
    - add / remove account
    - client deposit & withdraw
    - daily auto update of accounts balance
  - View activities – show logged details

© All rights reserved to John Bryce Training LTD

Bank

- details
    - clients
    - log service
    - account updater
- functionality:
    - get balance - this operation must calculate the balance each time the operation is called. The balance is calculated by summing the total client balance and the total accounts balance
    - add / remove client (effects bank total balance & should be logged)
    - get client list
    - view logs – allows to view activities
    - start account updater process

Client

- details:
    - id
    - rank (regular, gold, platinum)
    - name
    - balance
    - accounts
    - commission rate
    - interest rate
- functionality:
    - get methods for: id, name, balance, accounts
    - set methods for: name, balance
    - getFortune – returns the sum of client balance + total account balance
    - add account
    - remove account – money is transferred to the clients balance – no change in the bank total balance
    - deposit & withdraw – adds & removes money to clients balance – each action adds a commission to the bank total according to the following:
        - regular clients pays a commission rate of 3%
        - gold clients pays a commission rate of 2%
        - platinum clients pays a commission rate of 1%

© All rights reserved to John Bryce Training LTD

- update accounts balance (daily auto process)
  - regular clients gets a daily interest rate of 0.1%
  - gold clients gets a daily interest rate of 0.3%
  - platinum clients gets a daily interest rate of 0.5%

Account

- details:
  - id
  - balance
- functionality:
  - get methods for: id, balance
  - set methods for: balance

Log

- details:
  - time stamp
  - client-id
  - amount – the cash amount (+/-) of the operation
  - description – might by:
    - client added
    - client removed
    - client balance updated
      - deposit
      - withdraw
    - account updates
      - account closed
      - account opened
    - bank auto account update
- functionality:
  - print details

Logger

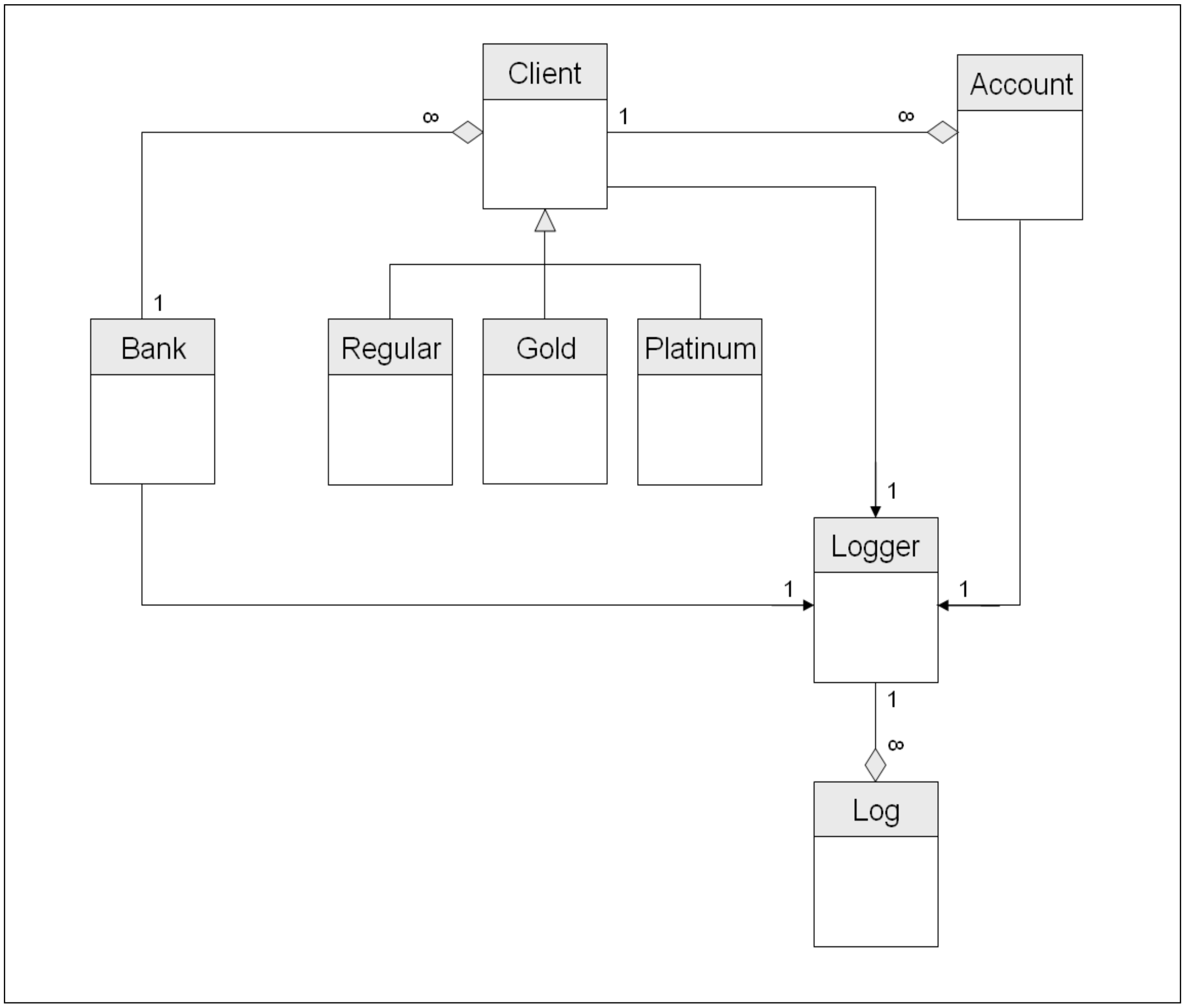
- details:
  - driver name
- functionality:

© All rights reserved to John Bryce Training LTD

- log – stores a new Log
- get logs – loads and returns all logs

Bank System

Phase 1 - System Design



© All rights reserved to John Bryce Training LTD

2.1. Module 6: encapsulation & basic flow control

Phase 2 – Building Classes

In this phase you'll create the building blocks of the bank system.
Later on the bank features will be added to those blocks.

In this phase the following classes will be defined:
Log
Logger
Account
Client
Bank

Classes details :

Log class

- Attributes:
    - timestamp - long
    - client-id - int
    - description – String
    - amount - float
- Methods:
    - Constructor (timestamp, client-id, description, amount)
    - getData() : String

Logger class

- Attributes:
    - driverName - String

© All rights reserved to John Bryce Training LTD

- Methods:
  - Constructor (driverName)
  - log(Log)   :   void   - implement to print Log on screen
  - *getLogs()   :   Log[]*   - leave empty for now

Account class

- Attributes:
  - id - int
  - balance - float
- Methods:
  - Constructor (id, balance)
  - get methods for: id, balance
  - set methods for: balance - log this operation

Client class

- Attributes:
  - id - int
  - name - String
  - balance - float
  - accounts – Account [5]
  - commission rate – float (value=0 for now)
  - interest rate – float (value=0 for now)
  - logger - Logger
- Methods:
  - Constructor (id, name, balance) – constructs a new client with zero accounts. Constructor also instantiates the Logger.
  - get methods for: id, name, balance, accounts
  - set methods for: name, balance
  - addAccount (Account)   : void - add the account to the array and log the operation.
    You should seek the array and place the Account where the first null value is found.
  - getAccount(int index) : Account – returns the account of the specified index or null if does not exist

© All rights reserved to John Bryce Training LTD

- remove account (int id) : void - remove the account with the same id from the array (by assigning a 'null' value to the array[position]) & transfers the money to the clients balance. Log the operation via creating Log object with appropriate data and sending it to the Logger.log(..) method.
- deposit(float) & withdraw(float) : void - implement to add of remove the amount from clients balance according to the commission (which is now zero). Use the commission data member in your calculation)
- autoUpdateAccounts() : void – run over the accounts, calculate the amount to add according to the client interest (meanwhile it is zero) and add it to each account balance. Use the interest data member in your calculation. Log this operation.
- getFortune() : float – returns the sum of client balance + total account balance.

Bank class

- Attributes:
  - clients – Client [100]
  - logService - Logger
  - account updater – leave this one for now
  - logger - Logger
- Methods:
  - An empty constructor that instantiates the clients array and logService.
  - setBalance() : void - this operation returns the bank balance. The balance is calculated by summing the total clients balance and the total accounts balance – you should use Client.getFortune() method of each client.
  - getBalance() : float – just returns bank current value.
  - addClient(Client) : void - add the client to the array and log the operation.
    You should seek the array and place the Client where the first null value is found.
  - removeClient(int id) : void - remove the client with the same id from the array (by assigning a 'null' value to the array[position]). Log the operation

© All rights reserved to John Bryce Training LTD

- getClients() : Client []
- view logs – prints all logs that are stored in the logger – leave empty for now
- *startAccountUpdater() : void* - leave empty for now

Note :

- Currently the timestamp value sent to the Log constructor should be zero.
- Add/Remove client methods – both affects the bank total balance – when a client is added – its balance is added to the bank balance. When a client is removed (this is done according to his id)– his balance and accounts balance are taken from the bank balance.
- Currently, "log the operation" means creating a Log object, filling it with the action details and print its getData() returned string value.

© All rights reserved to John Bryce Training LTD

2.2. Module 6-7: Inheritance & polymorphism

Phase 3 – System Core

In this phase you'll create the different types of Clients and implement more functionality in the bank system.

In this phase the following classes will be defined:
RegularClient
GoldClient
PlatinumClient

Bank class:

- Turn this class into a Single-ton
- Add a float static field commissionSum to sum all commissions due to client withdraw operation. Provide a static addCommission(float) method to update that field by adding commission every-time client perform withdraw
- Update setBalance() to add the commissionSum to the bank total balance
- Add a new method – printClientList() : void that prints the client details using the new toString() implementation you'll create in the next part.

Client class:

- Update Client to be an abstract class
- Change both interest & commission access modifiers to be protected instead of private
- Update withdraw() method to calculate commission according to the amount & commission rate and call addCommission() in Bank class to add the commission to the bank total.
- Override the equals() method perform the check according to the id value.
- Update the Bank.removeClient(id) to take a Client [Bank.removeClient(Client)] and perform the check using Client.equals(Object) method

© All rights reserved to John Bryce Training LTD

Sub-Classes details :

RegularClient, GoldClient, PlatinumClient

- are all extends Client
- adds:
  - commission rate – float with a fixed values:
    - regular clients pays a commission rate of 3%
    - gold clients pays a commission rate of 2%
    - platinum clients pays a commission rate of 1%
    - relevant for withdraw operation only
  - interest rate – float with a fixed values:
    - regular clients gets a daily interest rate of 0.1%
    - gold clients gets a daily interest rate of 0.3%
    - platinum clients gets a daily interest rate of 0.5%
    - later, a system thread will use it
- Note: all calculations should work fine since now the relevant interest & commission values are used.
- Override the toString() method to return the client type and ID

Account class:

- Override the equals() method to perform the check according to the id value.
- Update the Client.removeAccount(id) to take an Account [Client.removeAccount(Account)] and perform the check using Account.equals(Object) method

Logger class:

- Update log(Log) method to be static
- Update all log creators to use Logger.log(...) method

Log class

© All rights reserved to John Bryce Training LTD

- Override the toString() method to print log details (client ID, message & timestamp) – simply change the 'getData()' method name

2.3. Module 8: Exceptions

Phase 4 – Exceptions

In this phase you'll create a system exception indicates on problems when performing a withdraw operation

In this phase the following class will be defined:
WithdrawException

Class details :

- extends Exception
- Attribures:
    - clientId : int
    - currentBalance : float
    - withdrawAmount   : float
- Functionality:
- Constructor (message, clientId, currBalance, withdrawAmount)
- get methods for clientId, currBalance, withdrawAmount

Client class:

- Update the withdraw(float amount) method to throw WithdrawException if the amount to withdraw is greater than the current client balance.

© All rights reserved to John Bryce Training LTD

2.4. Module 9: Java utilities & Collections

Phase 5 – Adding Java Utilities

In this phase you'll update your system to work with Java Collections instead of static arrays.

Bank class:

- change the attribute clients to ArrayList
- update the addClient(..) and removeClient(..) methods accordingly

Client class:

- change the attribute accounts to ArrayList
- update the addAccount(..) and removeAccount(..) methods accordingly

Logger class:

- change the empty getLogs() method to return an ArrayList

Log class:

- Update Log toString() method to convert the timestamp from long into a java.util.Date

- Change all log generators to use a real timestamp and assign it to Log constructor (java.util.Date class)

© All rights reserved to John Bryce Training LTD

2.5. Module 10: Java Tiger syntax

Phase 6 – Updating code

In this phase you'll update your application to be type-safe and use auto-boxing

- Update all classes that use Java collection to use type-safe collections via generics
- Update all wrapper classes to auto-boxing new syntax

© All rights reserved to John Bryce Training LTD