# **Assignment 3**

Q1. Write a simple Banking System program by using OOPs concept where you can get account Holder name balance etc.?

1. **class** BankDetails {
2. **private** String accno;
3. **private** String name;
4. **private** String acc\_type;
5. **private** **long** balance;
6. Scanner sc = **new** Scanner(System.in);
7. //method to open new account
8. **public** **void** openAccount() {
9. System.out.print("Enter Account No: ");
10. accno = sc.next();
11. System.out.print("Enter Account type: ");
12. acc\_type = sc.next();
13. System.out.print("Enter Name: ");
14. name = sc.next();
15. System.out.print("Enter Balance: ");
16. balance = sc.nextLong();
17. }
18. //method to display account details
19. **public** **void** showAccount() {
20. System.out.println("Name of account holder: " + name);
21. System.out.println("Account no.: " + accno);
22. System.out.println("Account type: " + acc\_type);
23. System.out.println("Balance: " + balance);
24. }
25. //method to deposit money
26. **public** **void** deposit() {
27. **long** amt;
28. System.out.println("Enter the amount you want to deposit: ");
29. amt = sc.nextLong();
30. balance = balance + amt;
31. }
32. //method to withdraw money
33. **public** **void** withdrawal() {
34. **long** amt;
35. System.out.println("Enter the amount you want to withdraw: ");
36. amt = sc.nextLong();
37. **if** (balance >= amt) {
38. balance = balance - amt;
39. System.out.println("Balance after withdrawal: " + balance);
40. } **else** {
41. System.out.println("Your balance is less than " + amt + "\tTransaction failed...!!" );
42. }
43. }
44. //method to search an account number
45. **public** **boolean** search(String ac\_no) {
46. **if** (accno.equals(ac\_no)) {
47. showAccount();
48. **return** (**true**);
49. }
50. **return** (**false**);
51. }
52. }
53. **public** **class** BankingApp {
54. **public** **static** **void** main(String arg[]) {
55. Scanner sc = **new** Scanner(System.in);
56. //create initial accounts
57. System.out.print("How many number of customers do you want to input? ");
58. **int** n = sc.nextInt();
59. BankDetails C[] = **new** BankDetails[n];
60. **for** (**int** i = 0; i < C.length; i++) {
61. C[i] = **new** BankDetails();
62. C[i].openAccount();
63. }
64. // loop runs until number 5 is not pressed to exit
65. **int** ch;
66. **do** {
67. System.out.println("\n \*\*\*Banking System Application\*\*\*");
68. System.out.println("1. Display all account details \n 2. Search by Account number\n 3. Deposit the amount \n 4. Withdraw the amount \n 5.Exit ");
69. System.out.println("Enter your choice: ");
70. ch = sc.nextInt();
71. **switch** (ch) {
72. **case** 1:
73. **for** (**int** i = 0; i < C.length; i++) {
74. C[i].showAccount();
75. }
76. **break**;
77. **case** 2:
78. System.out.print("Enter account no. you want to search: ");
79. String ac\_no = sc.next();
80. **boolean** found = **false**;
81. **for** (**int** i = 0; i < C.length; i++) {
82. found = C[i].search(ac\_no);
83. **if** (found) {
84. **break**;
85. }
86. }
87. **if** (!found) {
88. System.out.println("Search failed! Account doesn't exist..!!");
89. }
90. **break**;
91. **case** 3:
92. System.out.print("Enter Account no. : ");
93. ac\_no = sc.next();
94. found = **false**;
95. **for** (**int** i = 0; i < C.length; i++) {
96. found = C[i].search(ac\_no);
97. **if** (found) {
98. C[i].deposit();
99. **break**;
100. }
101. }
102. **if** (!found) {
103. System.out.println("Search failed! Account doesn't exist..!!");
104. }
105. **break**;
106. **case** 4:
107. System.out.print("Enter Account No : ");
108. ac\_no = sc.next();
109. found = **false**;
110. **for** (**int** i = 0; i < C.length; i++) {
111. found = C[i].search(ac\_no);
112. **if** (found) {
113. C[i].withdrawal();
114. **break**;
115. }
116. }
117. **if** (!found) {
118. System.out.println("Search failed! Account doesn't exist..!!");
119. }
120. **break**;
121. **case** 5:
122. System.out.println("See you soon...");
123. **break**;
124. }
125. }
126. **while** (ch != 5);
127. }
128. }

Q2. Write a Program where you inherit method from parent class and show method Overridden Concept?

class Grandpa {

public void show()

{

System.out.println("Inside show() method of Grandpa class");

}

}

class Dad extends Grandpa {

@Override public void show()

{

System.out.println("Inside show() method of Dad class");

}

class Me extends Dad {

@Override public void show()

{

System.out.println("Inside show() method of Me class");

}

}

public class Test {

public static void main(String[] args)

{

Grandpa grandpa = new Grandpa();

Grandpa dad = new Dad();

Grandpa me = new Me();

grandpa.show();

dad.show();

me.show() } }

Q3. Write a program to show run time polymorphism in java?

## Java Runtime Polymorphism Example: Bank

Consider a scenario where Bank is a class that provides a method to get the rate of interest. However, the rate of interest may differ according to banks. For example, SBI, ICICI, and AXIS banks are providing 8.4%, 7.3%, and 9.7% rate of interest.

**class** Bank{

**float** getRateOfInterest(){**return** 0;}

}

**class** SBI **extends** Bank{

**float** getRateOfInterest(){**return** 8.4f;}

}

**class** ICICI **extends** Bank{

**float** getRateOfInterest(){**return** 7.3f;}

}

**class** AXIS **extends** Bank{

**float** getRateOfInterest(){**return** 9.7f;}

}

**class** TestPolymorphism{

**public** **static** **void** main(String args[]){

Bank b;

b=**new** SBI();

System.out.println("SBI Rate of Interest: "+b.getRateOfInterest());

b=**new** ICICI();

System.out.println("ICICI Rate of Interest: "+b.getRateOfInterest());

b=**new** AXIS();

System.out.println("AXIS Rate of Interest: "+b.getRateOfInterest());

}

}

Q4. Write a program to show Compile time polymorphism in java?

// overloading by changing the number of parameters

public class MethodOverloading {

void show(int num1)

{

System.out.println("number 1 : " + num1);

}

void show(int num1, int num2)

{

System.out.println("number 1 : " + num1+ " number 2 : " + num2);

}

public static void main(String[] args)

{

MethodOverloading obj = new MethodOverloading();

obj.show(3);

obj.show(4, 5);

}

}

Q5. Achieve loose coupling in java by using OOPs concept?

To achieve loose coupling, one should use abstract classes or interface while performing inheritance. The following examples will make the concept clearer.

**class** A

{

**void** foo()

{

System.out.println("Inside the foo method of base class.");

}

}

// child or derived class

**class** B **extends** A

{

**void** foo()

{

System.out.println("Inside the foo method of derived class.");

}

}

**public** **class** CouplingExample

{

// main method

**public** **static** **void** main(String argvs[])

{

B obj = **new** B();

obj.foo();

}

}

However, there is a problem with the above code. Inheritance has led to the tight coupling of classes A and B. Class B knows a lot of stuff about class A. Also, there are fair chances that changes in class A might impact class B. Let's modify the above code to understand it.

### Loose Coupling Code

**interface** Parent

{

**void** foo();

}

**class** A **implements** Parent

{

// parameterized constructor

A(**int** x, **int** y)

{

}

**public** **void** foo()

{

System.out.println("In the foo method of class A.");

}

}

**class** B **implements** Parent

{

**public** **void** foo()

{

System.out.println("In the foo method of class B.");

}

}

**public** **class** CouplingExample2

{

**public** **static** **void** main(String args[])

{

B obj = **new** B();

obj.foo();

}

}

## Q6. What is the benefit of encapsulation in java?

## Benefits of Encapsulation in Java

* A class can have complete control over its data members and data methods.
* The class will maintain its data members and methods as read-only.
* Data hiding prevents the user from the complex implementations in the code.
* The variables of the class can be read-only or write-only as per the programmer's requirement.
* Encapsulation in Java provides an option of code-reusability.
* Using encapsulation will help in making changes to an existing code quickly.
* Unit testing a code designed using encapsulation is elementary.
* Standard IDEs have the support of getters and setters; this makes coding even faster.

Q7. Is java a 100% Object oriented Programming language? If not why?

Pure Object Oriented Language or Complete Object Oriented Language are Fully Object Oriented Language which supports or have features which treats everything inside program as objects. It doesn’t support primitive datatype(like int, char, float, bool, etc.). There are seven qualities to be satisfied for a programming language to be pure Object Oriented. They are:

Encapsulation/Data Hiding

Inheritance

Polymorphism

Abstraction

All predefined types are objects

All user defined types are objects

All operations performed on objects must be only through methods exposed at the objects.

**Why Java is not a Pure Object Oriented Language?**

Java supports property 1, 2, 3, 4 and 6 but fails to support property 5 and 7 given above. Java language is not a Pure Object Oriented Language as it contains these properties:

* **Primitive Data Type ex. int, long, bool, float, char, etc. as Objects:** Smalltalk is a “pure” object-oriented programming language unlike Java and C++ as there is no difference between values which are objects and values which are primitive types. In Smalltalk, primitive values such as integers, booleans and characters are also objects. In Java, we have predefined types as non-objects (primitive types).

int a = 5;   
System.out.print(a);

* **The static keyword:** When we declares a class as static then it can be used without the use of an object in Java. If we are using static function or static variable then we can’t call that function or variable by using dot(.) or class object defying object oriented feature.
* **Wrapper Class:** Wrapper class provides the mechanism to convert primitive into object and object into primitive. In Java, you can use Integer, Float etc. instead of int, float etc. We can communicate with objects without calling their methods. ex. using arithmetic operators.

String s1 = "ABC" + "A" ;

* Even using Wrapper classes does not make Java a pure OOP language, as internally it will use the operations like Unboxing and Autoboxing. So if you create Integer instead of int and do any mathematical operation on it, under the hoods Java is going to use primitive type int only.

public class BoxingExample

{

public static void main(String[] args)

{

Integer i = new Integer(10);

Integer j = new Integer(20);

Integer k = new Integer(i.intValue() + j.intValue());

System.out.println("Output: "+ k);

}

}

* **In the above code, there are 2 problems where Java fails to work as pure OOP:**

1. While creating Integer class you are using primitive type “int” i.e., numbers 10, 20.

2.While doing addition Java is using primitive type “int”.

Q8. What are the advantages of abstraction in java?

## Advantages of Abstract Class

### Template

The abstract class in Java enables the best way to execute the process of data abstraction by providing the developers with the option of hiding the code implementation. It also presents the end-user with a template that explains the methods involved.

### Loose Coupling

Data abstraction in Java enables loose coupling, by reducing the dependencies at an exponential level.

### Code Reusability

Using an abstract class in the code saves time. We can call the abstract method wherever the method is necessary. Abstract class avoids the process of writing the same code again.

### Abstraction

Data abstraction in Java helps the developers hide the code complications from the end-user by reducing the project's complete characteristics to only the necessary components.

### Dynamic Resolution

Using the support of dynamic method resolution, developers can solve multiple problems with the help of one abstract method.

Q9. What is an abstraction explained with an Example?

**Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

Another way, it shows only essential things to the user and hides the internal details, for example, sending SMS where you type the text and send the message. You don't know the internal processing about the message delivery.

### Ways to achieve Abstraction

There are two ways to achieve abstraction in java

1 Abstract class (0 to 100%)

2 Interface (100%)

Example: Bank

**abstract** **class** Bank{

**abstract** **int** getRateOfInterest();

}

**class** SBI **extends** Bank{

**int** getRateOfInterest(){**return** 7;}

}

**class** PNB **extends** Bank{

**int** getRateOfInterest(){**return** 8;}

}

**class** TestBank{

**public** **static** **void** main(String args[]){

Bank b;

b=**new** SBI();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

b=**new** PNB();

System.out.println("Rate of Interest is: "+b.getRateOfInterest()+" %");

}}

Q10. What is the final class in Java?

The final modifier for finalizing the implementations of classes, methods, and variables.

The main purpose of using a class being declared as final is to prevent the class from being subclassed. If a class is marked as final then no class can inherit any feature from the final class.

You cannot extend a final class. If you try it gives you a compile time error.

## Example

final class Super {  
 private int data = 30;  
}  
public class Sub extends Super{  
 public static void main(String args[]){  
 }  
}

## Output

Exception in thread "main" java.lang.Error: Unresolved compilation problem:  
 at newJavaExamples.Sub.main(Sub.java:9)