# Object Oriented Programming (Java)
# March 2023

Q.NO.1

a. For each of the following java expressions, provide the data type and thevalue. Use the appropriate java literal to express the value.

| Expression | Type | Value |
|---|---|---|
| 1 + 2 | int | 3 |
| 100 * .015 | double | 1.5 |
| true & false | boolean | true |
| “” + 3 + 2 | String | 32 |
| ½ + ½ | int | 0 |

Ans:-

| Expression | Type | Value |
|---|---|---|
| 1 + 2 | int | 3 |
| 100 * .015 | double | 1.5 |
| true & false | boolean | false |
| “” + 3 + 2 | String | “32” |
| ½ + ½ | double | 1.0 |

b. Explain the differences between an if statement and a while statement.

| Feature | `if` Statement | `while` Statement |
|---|---|---|
| Purpose | Used for conditional branching in a program. | Used for repeated execution based on a condition. |
| Syntax | `if (condition) { // code block }` | `while (condition) { // code block }` |
| Execution | Executes the code block only if the condition is true. | Executes the code block repeatedly as long as the condition is true. |
| Frequency | Executes the code block at most once. | Executes the code block multiple times (possibly zero). |
| Conditional | Can be followed by an `else` statement for an alternative code block when the condition is false. | Doesn't have a direct "else" alternative; you might use `do-while` for similar functionality. |
| Loop Control | Doesn't inherently control looping; used for branching. | Controls looping based on the condition. |
| Example | `java if (x > 10) { // do something }` | `java while (count < 10) { // do something count++; }` |

c. Write a switch statement to check the character grade and show the Message based on the table below:

| Grade | Message |
|---|---|
| A | Excellent |
| B, C | Well Done |
| D | You Passed |
| F | Try Again |
| Other Grades | Invalid Grade |

Ans:-

```java
public class GradeMessageSwitch {
    public static void main(String[] args) {
        char grade = 'A';

        String message;

        switch (grade) {
            case 'A':
                message = "Excellent";
                break;
            case 'B':
            case 'C':
                message = "Well Done";
                break;
            case 'D':
                message = "You Passed";
                break;
            case 'F':
                message = "Try Again";
                break;
            default:
                message = "Invalid Grade";
                break;
        }

        System.out.println("Grade: " + grade);
        System.out.println("Message: " + message);
    }
}
```

d. Convert the following while loop to for

```
loop:int sum = 0;
int count = 3;
while (count<=
        1000){Sum +=
        count; Count
        += 3;
}
System.out.println("Sum: " + sum);
```

Ans:-

```
int sum = 0;
for (int count = 3; count <= 1000; count
        +=3){Sum += count;
}
System.out.println("Sum: " + sum);
```

e. i. Write the java code to declare a variable of type string and initialize it to"Whatever".

Ans:

```
-   Public class StringExample {
        Public static void main(String[]
            args) {String myString =
            "Whatever";
            System.out.println("My String:" + myString);
        }
    }
```

ii. What will be the output of the following

```
code?public class FinalExam1 {
        public static void main(String[]
                args){String S2 = “is taught”;
                Boolean bEquals = S2.equals(“is
                taught”);System.out.println(bEquals);
        }
}
```

Ans:- true

Q.No. 2

a. Write a java class called CreditCard with the following description:

Instance variables/data members:

- name (String)
- balance (double)
- limit

(double)Member methods:

- default constructor
- parameterized constructor
- getName()
- getBalance()
- chargeCard (double amount)
- makepayment (double amount)
- toString(

)Note:

- The first constructor should initialize name to null and balance to 0.
- The second constructor initializes name and balance to the parameterspassed.
- chargeCard increase the balance.
- makePayment decreases the balance.

- toString() method returns card holder name and the current balance separated by a comma; for example if name is jake and balance is 40.0 it should return jake, RM40.00.

Ans:-

```java
import java.text.DecimalFormat;

public class CreditCard {
    private String name;
    private double balance;
    private double limit;

    public CreditCard() {
        this.name = null;
        this.balance = 0.0;
        this.limit = 0.0;
    }

    public CreditCard(String name, double balance, double limit) {
        this.name = name;
        this.balance = balance;
        this.limit = limit;
    }

    public String getName() {
        return name;
    }

    public double getBalance() {
        return balance;
    }

    public void chargeCard(double amount) {
        if (amount > 0) {
            balance += amount;
        } else {
            System.out.println("Invalid charge amount.");
        }
    }
```

```
35
36      public void makePayment(double amount) {
37          if (amount > 0) {
38              if (amount <= balance) {
39                  balance -= amount;
40              } else {
41                  System.out.println("Payment amount exceeds balance.");
42              }
43          } else {
44              System.out.println("Invalid payment amount.");
45          }
46      }
47
48      @Override
49      public String toString() {
50          DecimalFormat df = new DecimalFormat("RM0.00");
51          return name + ", " + df.format(balance);
52      }
53
54      public static void main(String[] args) {
55          CreditCard card1 = new CreditCard("Jake", 40.0, 1000.0);
56          System.out.println(card1);
57
58          card1.chargeCard(50.0);
59          System.out.println(card1);
60
61          card1.makePayment(30.0);
62          System.out.println(card1);
63      }
64  }
65
```

b. Write a driver program called CreditCardApp that

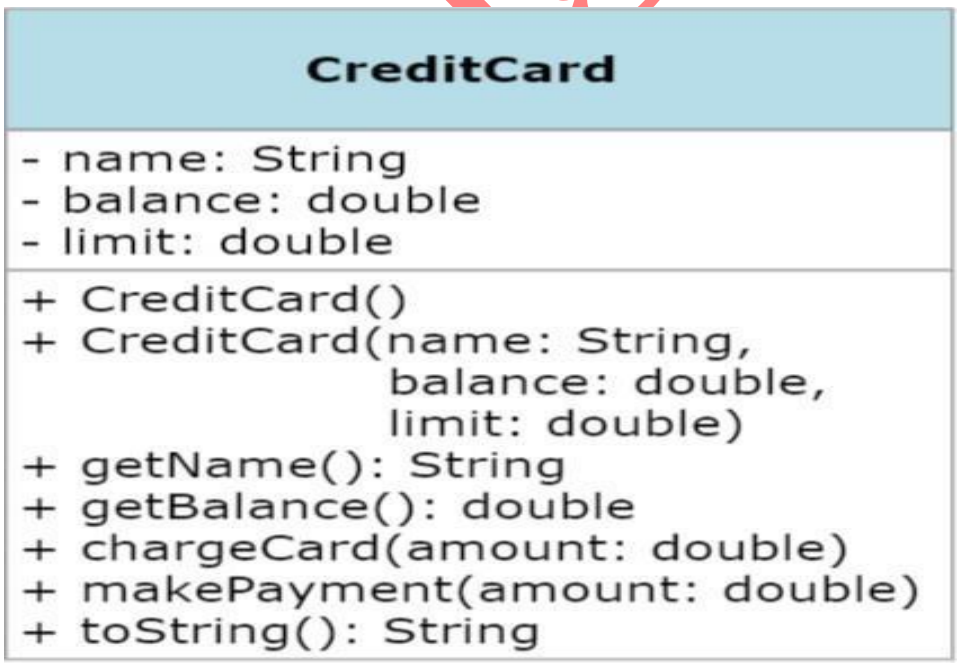
- creates a CreditCard object called c1 and assigns “John” to its name fieldand 1000 to balance field.
- Calls the chargeCard method with 500 to increase balance
- Calls the makePayment method with 300 to decreases the balance
- Calls the getBalance method to display the name and current balance
- Creates another object called c2 without passing parameters and displaythe name and current balance for this object.

Ans:-

```java
public class CreditCardApp {
    public static void main(String[] args) {
        CreditCard c1 = new CreditCard("John", 1000.0, 2000.0);

        c1.chargeCard(500.0);
        c1.makePayment(300.0);

        System.out.println(c1.getName() + "'s current balance: RM" + c1.getBalance());

        CreditCard c2 = new CreditCard();
        System.out.println(c2.getName() + "'s current balance: RM" + c2.getBalance());
    }
}
```

c. Draw a UML class diagram for the CreditCard class in

Ans:-



Q.NO. 3

a. Write a JAVA method that takes an array of integers as a parameter and uses a for loop to find and return the minimum value integer in the array. Just write the method, not a whole class or program.

Ans:-

```java
public static int findMinimum(int[] array) {
    if (array == null || array.length == 0) {
        throw new IllegalArgumentException("Array is empty or null");
    }

    int min = array[0];

    for (int i = 1; i < array.length; i++) {
        if (array[i] < min) {
            min = array[i];
        }
    }

    return min;
}
```

b. i. Explain the difference between accessor (get) methods and mutator (set)methods.

Ans:-

| Aspect | Accessor (Get) Methods | Mutator (Set) Methods |
|---|---|---|
| Purpose | Retrieve the value of an attribute or property. | Modify the value of an attribute or property. |
| Naming Convention | Typically named using the "get" prefix followed by the attribute name. (e.g., `` `getName()` ``) | Typically named using the "set" prefix followed by the attribute name. (e.g., `` `setName(String name)` ``) |
| Return Value | Returns the current value of the attribute. | Typically returns `` `void` ``, as they modify the attribute directly. |
| Parameters | Usually no parameters or minimal (e.g., `` `getId()` ``). | Accepts one or more parameters to set the new value. |
| Impact on Object | Does not modify the object's internal state. | Modifies the object's internal state. |

ii. Give one example of method declaration for mutator method and one example of method declaration for accessor method in Java.

Ans:-

```
Mutator (Set) Method Example:

public void setName(String newName) {
    this.name = newName;
}


Accessor (Get) Method Example:

public int getAge() {
    return this.age;
}
```

c. Explain what is wrong with the following overloading methods. If you run theJava Compiler, what message are you likely to get?

```
public class Example {
        public int sum(int a, int b);
        public double sum(int a, int b);
}
```

Ans:-

1. **Missing Method Body:** Both method declarations lack method bodies, which means there's no implementation provided for these methods. In Java, when you declare a method in a class, you need to provide an implementation (method body) for it.

If you attempt to compile this code, you are likely to encounter the following error messages:

csharp Copy code

```
Example.java:2: error: missing method body, or declare abstract
    public int sum(int a, int b);
```

csharp Copy code

```
Example.java:3: error: missing method body, or declare abstract
    public double sum(int a, int b);
```

d. Consider the following scenario:
An Employee Management System has to be created for a factory. The requirements are as given below:

- All employee should report at the security to log the time when thecome to duty and also log the out time when the leave factory.
- The security persons should check the register by end of the day andcheck if any employee has not logged his time and should mark absent.
- The accountant should calculate the number of days an employee hasworked and should credit the salary to employee's bank account.
- The employee will have an employee id, name, department, bank account number etc.

The system should ask for an employee id, his in time and out time of a dayand should give the details of number days an employee has worked.

Identify:

i. Any Two classes
ii. Any Two possible attributes for each class; and
iii. Any One operation for each class in the above scenario.

Ans:-

**i. Classes:**

1. Employee:
This class represents the information related to an employee.
2. AttendanceRecord:
This class represents the attendance record of an employee for a specific day.

**ii. Attributes**

Employee Class:
1. employeeId: An identifier for each employee.
2. name: The name of the employee.
3. department: The department in which the employee works.
4. bankAccountNumber: The bank account number for salary deposit.

AttendanceRecord Class:

1. employeeId: The identifier of the employee for whom the attendance recordis being stored.
2. inTime: The time when the employee logs in.
3. outTime: The time when the employee logs out.
4. date: The date for which the attendance record is being stored.

**iii. Operations**

Employee Class:

1. calculateSalary(): This operation could calculate the salary for the employee based on their attendance records and other factors. It involves interactingwith the accountant class to credit the salary.

AttendanceRecord Class:

1. calculateWorkDuration(): This operation calculates the duration of work for an employee on a particular day by subtracting the in time from the out time. This information can be used to determine the number of days an employee has worked.

**Q.No.4**

a.

i. What are two main reasons for using files?

Ans:- Two main reasons for using files:

**1. Data Persistence:** Files are used to store data persistently on storage devices such as hard drives, solid-state drives, or external storage media. This allows data to be saved even after a program exits or the computer is turned off, ensuring that information is retained for future use.

**2. Data Sharing:** Files serve as a common medium for sharing data between different programs and systems. By writing data to files, one program can pass information to another, and files can be easily transferred over networks, allowing for interoperability and data exchange.

ii. Give any two reasons for java.io.FileNotFoundExccception to be thrown at run-time.

Ans:- Two reasons for `java.io.FileNotFoundException` to be thrown at runtime:

1. **File Not Found:** This exception is thrown when an attempt is made to access a file that does not exist at the specified file path. This can happen if the file was deleted, moved, or if the path provided to open the file is incorrect.
2. **Insufficient Permissions:** Another reason for `FileNotFoundException` is insufficient permissions to access the file. If the program does not have the necessary read permissions for the file or if the file is protected by the operating system's security settings, this exception will be thrown when attempting to access the file.

b. Write in java that reads a file named 'infile.txt' and copies it to a file called 'outfile.txt'.

Ans:-

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class FileCopy {
    public static void main(String[] args) {
        String inputFile = "infile.txt";
        String outputFile = "outfile.txt";

        try {
            BufferedReader reader = new BufferedReader(new FileReader(inputFile));
            BufferedWriter writer = new BufferedWriter(new FileWriter(outputFile));

            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine();
            }

            reader.close();
            writer.close();

            System.out.println("File copy complete.");
        } catch (IOException e) {
            e.printStackTrace();
            System.err.println("An error occurred: " + e.getMessage());
        }
    }
}
```

c. A programmer was advised to decompose a large module into multiple sub modules.

i. Explain any two reasons for this.

Ans:-

**Improved Maintainability**: Breaking a large module into multiple sub-modules improves code maintainability for several reasons. First, it makes the codebase more organized and easier to understand. Developers can focus on smaller, more manageable pieces of code, which reduces the cognitive load when working on the project. Additionally, when a bug or issue arises, it is often easier to isolate and fix the problem within a smaller sub-module than within a large, monolithic module. This modularity also facilitates code reusability because sub-modules can be used in multiple parts of the project or even in other projects, leading to a more efficient development process.

**Parallel Development:** Decomposing a large module into sub-modules allows multiple developers or teams to work concurrently on different parts of the project. Each sub-module can be assigned to a different developer or team, reducing bottlenecks and speeding up the development process. This parallel development approach can lead to faster project completion and better utilization of development resources.

ii. Explain how java support modular programming

Ans:- Java introduced significant support for modular programming with the release of Java 9 through the introduction of the Java Platform Module System (JPMS), also known as Project Jigsaw. The JPMS provides tools and features for creating modular applications in Java.

Here's how Java supports modular programming:

**1. Modules:** Java introduced the concept of modules, which allow developers to encapsulate and organize code into discrete units. A module is a collection of related packages, classes, and resources that can specify which parts of its code are accessible to other modules and which are private. This promotes strong encapsulation and helps avoid classpath conflicts.

**2. Module Declarations:** A module is defined by a module-info.java file, which declares the module's name, dependencies on other modules, and what it exports (makes available) to other modules. This declaration provides clear boundaries and dependencies between modules, making it easier to manage complex software projects.

**3. Encapsulation:** Modules allow you to control the visibility of classes, methods, and other elements within a module. You can specify which classes are accessible outside the module (public API) and which are for internal use only. This encapsulation helps prevent unintended access and reduces potential issues caused by changes to internal implementations.

**4. Improved Dependency Management:** With modules, Java provides a more explicit way to declare and manage dependencies between different parts of your application. This helps ensure that the required modules are available at runtime and allows for more efficient packaging and distribution of Java applications.

**5. Modular JAR Files:** Java 9 introduced the concept of modular JAR files, which are JAR files containing module-info.class files. These modular JAR files make it easier to distribute and deploy modular Java applications.

**6. jlink Tool:** The jlink tool allows you to create custom runtime images containing only the modules your application requires. This reduces the size of the runtime environment and helps eliminate unused code, resulting in more efficient and optimized deployments.