Question 1

a) List the four (4) basic features of OOP. (4 marks)

1. Encapsulation: The bundling of data and methods that operate on the data within a single unit or class. It restricts direct access to some of an object's components, which can prevent the accidental modification of data.
2. Inheritance: The mechanism by which one class can inherit the properties and methods of another class. This allows for the creation of a new class based on an existing class, promoting code reuse and the establishment of a hierarchical relationship between classes.
3. Polymorphism: The ability of different objects to respond to the same method call in different ways. It allows for one interface to be used for a general class of actions, with the specific action determined by the exact nature of the situation.
4. Abstraction: The concept of hiding the complex implementation details of a system and exposing only the necessary and relevant features. This helps in reducing complexity and allowing the programmer to focus on interactions at a higher level.

b) List five (5) advantages that OOP provides. (5 marks)

1. Modularity: Code can be divided into smaller, manageable, and interchangeable pieces (classes), which makes it easier to troubleshoot, test, and maintain.
2. Reusability: Through inheritance and polymorphism, OOP allows for the reuse of existing code, reducing redundancy and the amount of effort needed to write new code from scratch.
3. Scalability: OOP allows for the easy addition of new features and the expansion of programs without significant changes to the existing codebase.
4. Maintainability: Encapsulation ensures that the implementation details of a class are hidden from the outside world, which means changes to one part of the system have minimal impact on other parts.
5. Improved Productivity: By using pre-defined libraries and frameworks, OOP can significantly speed up the development process, allowing developers to focus on solving the specific problems of their applications.

c) Give two (2) differences between a Class and an Object. (4 marks)

1. Definition:
    - Class: A blueprint or template for creating objects. It defines a type of object according to the data it holds and the operations that can be performed on that data.
    - Object: An instance of a class. It is a concrete realization of the class that can be used to perform operations defined by the class.
2. Existence:
    - Class: Exists as a logical construct in the code. It defines the properties and behaviors that its objects will have but does not occupy memory until an object is created.
    - Object: Exists in memory during the runtime of the program. It holds actual data and can perform actions as defined by the class.

d) Differentiate between public, private and protected access modifiers. (6 marks)

1. Public:
    - Access Level: Members declared as public are accessible from any other class in the program.
    - Use Case: Used when the member needs to be accessible from anywhere, such as utility methods or constants.
2. Private:
    - Access Level: Members declared as private are accessible only within the class in which they are declared.
    - Use Case: Used to hide implementation details and restrict access to the internal state of the object, ensuring encapsulation.
3. Protected:
    - Access Level: Members declared as protected are accessible within the same package and by subclasses, even if they are in different packages.
    - Use Case: Used when a member needs to be accessed by subclasses or classes within the same package but not by the general public, promoting inheritance while maintaining some level of encapsulation.

e) Explain each type of Inheritance mentioned below:

i. Single inheritance (3 marks)

Single inheritance is a type of inheritance in which a class (called a subclass or derived class) inherits the properties and behaviors of a single parent class (called a superclass or base class). It establishes a one-to-one relationship between the child and parent class, allowing the child class to use and extend the functionalities of the parent class.

Example:

```
class Animal {
    void eat() {
        System.out.println("Eating...");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking...");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();  // Inherited from Animal
        d.bark(); // Defined in Dog
    }
}
```

ii. Multiple inheritance (3 marks)

Multiple inheritance is a type of inheritance in which a class can inherit properties and behaviors from more than one parent class. This allows for the combination of functionalities from multiple classes into a single class. However, Java does not support multiple inheritance directly through classes to avoid complexity and ambiguity (the diamond problem). Instead, Java supports multiple inheritance through interfaces.

Example:

```java
interface Animal {
    void eat();
}

interface Pet {
    void play();
}

class Dog implements Animal, Pet {
    public void eat() {
        System.out.println("Eating...");
    }

    public void play() {
        System.out.println("Playing...");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog d = new Dog();
        d.eat();  // From Animal interface
        d.play(); // From Pet interface
    }
}
```

Question 2

a) Polymorphism is a concept where objects of different types can be accessed through the same interface. Each type can provide its own independent implementation of this interface. Using the class X below, mention the other name and provide a Java example of:

i. Static polymorphism (5 marks)

Other Name: Compile-time polymorphism

Example:

```
class X {
    // Method overloading
    public int sum(int a, int b) {
        return a + b;
    }

    public int sum(int a, int b, int c) {
        return a + b + c;
    }

    public static void main(String[] args) {
        X obj = new X();
        System.out.println("Sum of two numbers: " + obj.sum(10, 20));
        System.out.println("Sum of three numbers: " + obj.sum(10, 20, 30));
    }
}
```

ii. Dynamic polymorphism (5 marks)

Other Name: Run-time polymorphism

Example:

```
class X {
    public void sum() {
        System.out.println("Class X sum method");
    }
}

class Y extends X {
    public void sum() {
        System.out.println("Class Y sum method");
    }
}

public class Main {
    public static void main(String[] args) {
        X obj = new Y(); // Polymorphic behavior
        obj.sum(); // This will call the sum method in class Y
    }
}
```

b) Class SearchEngine, below, provides a search bar in a GUI. The GUI is displayed properly, but clicking the search button does nothing. Your task is to make changes to SearchEngine class so that it will listen for a click of the search button and call method search with the appropriate text if that event occurs. (10 marks)

```java
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class SearchEngine extends JFrame {
    private JTextField searchBar = new JTextField("Enter your search here");
    private JButton submit = new JButton("Search");

    public SearchEngine() {
        Container cp = getContentPane();
        setSize(300, 100);
        setResizable(false);
        cp.add(searchBar, BorderLayout.CENTER);
        cp.add(submit, BorderLayout.WEST);
        setVisible(true);
        pack();

        // Add action listener to the button
        submit.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                search(searchBar.getText());
            }
        });
    }

    private void search(String input) {
        // Method implementation
        System.out.println("Searching for: " + input);
    }

    public static void main(String[] args) {
        new SearchEngine();
    }
}
```

c) For each of the following operations, indicate the worst-case run time for a list of size n. Your answer should be written in Big-O notation. (5 marks)

i. Finding the size of a singly linked list without a size field

Answer: O(n)

ii. Finding the size of a doubly linked list without a size field

Answer: O(n)

iii. Getting the last element of a doubly linked list

Answer: O(1)

iv. Finding a value in a sorted doubly linked list

Answer: O(n) (for linear search) or O(log n) (for binary search if the list is converted to an array first)

v. Getting an element at a particular index of singly linked list

Answer: O(n)

Question 3

a) Write a Java program to create a base class Person and declare two (2) data members. (5 marks)

```java
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

b) Use the class created to demonstrate inheritance. In order to do this, derive a class called Staff from class Person and then another class called Admin from class Person. Give two (2) additional data members, constructor(s) and member functions as necessary. (15 marks)

```
class Staff extends Person {
    String position;
    double salary;

    Staff(String name, int age, String position, double salary) {
        super(name, age);
        this.position = position;
        this.salary = salary;
    }

    void display() {
        super.display();
        System.out.println("Position: " + position);
        System.out.println("Salary: " + salary);
    }
}

class Admin extends Person {
    String department;
    int experience;

    Admin(String name, int age, String department, int experience) {
        super(name, age);
        this.department = department;
        this.experience = experience;
    }

    void display() {
        super.display();
        System.out.println("Department: " + department);
        System.out.println("Experience: " + experience + " years");
    }
}
```

c) Use the Main() method to demonstrate object creation, access to data members and invoking member functions. (5 marks)

```
public class Main {
    public static void main(String[] args) {
        Staff staffMember = new Staff("Alice", 30, "Manager", 75000.00);
        Admin adminMember = new Admin("Bob", 45, "IT", 20);

        System.out.println("Staff Member Details:");
        staffMember.display();

        System.out.println("\nAdmin Member Details:");
        adminMember.display();
    }
}
```

Complete Program:

```
class Person {
    String name;
    int age;

    Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    void display() {
        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}

class Staff extends Person {
    String position;
    double salary;

    Staff(String name, int age, String position, double salary) {
        super(name, age);
        this.position = position;
        this.salary = salary;
    }

    void display() {
        super.display();
        System.out.println("Position: " + position);
        System.out.println("Salary: " + salary);
    }
}

class Admin extends Person {
    String department;
    int experience;

    Admin(String name, int age, String department, int experience) {
        super(name, age);
        this.department = department;
```

```
        this.experience = experience;
    }

    void display() {
        super.display();
        System.out.println("Department: " + department);
        System.out.println("Experience: " + experience + " years");
    }
}

public class Main {
    public static void main(String[] args) {
        Staff staffMember = new Staff("Alice", 30, "Manager", 75000.00);
        Admin adminMember = new Admin("Bob", 45, "IT", 20);

        System.out.println("Staff Member Details:");
        staffMember.display();

        System.out.println("\nAdmin Member Details:");
        adminMember.display();
    }
}
```

In this program:

- The `Person` class is the base class with `name` and `age` data members.
- The `Staff` class extends `Person` and adds `position` and `salary` data members.
- The `Admin` class extends `Person` and adds `department` and `experience` data members.
- The `Main` class demonstrates object creation, accessing data members, and invoking member functions.

Question 4

a) What is an Exception? (3 marks)

An exception is an event that disrupts the normal flow of a program's execution. It is an object which is thrown at runtime when an unexpected condition arises, such as an attempt to divide by zero, a file not found, or an array index out of bounds. Exceptions provide a way to handle errors or other exceptional events in a controlled manner.

b) Extend the code shown below to handle the Exception that can occur when opening a file that does not exist. You do not have to add the import that is needed for the exception. (5 marks)

```
public void readFile(String filename) {
    File file = new File(filename);
    Scanner scan = null;
    try {
        scan = new Scanner(file);
        // Add your file reading logic here
    } catch (FileNotFoundException e) {
        System.out.println("File not found: " + filename);
    } finally {
        if (scan != null) {
            scan.close();
        }
    }
}
```

c) What happens if several Catch blocks match the type of the thrown object? (3 marks)

If several catch blocks match the type of the thrown exception, the first matching catch block that appears in the order will be executed. Once a matching catch block is found and executed, the control exits the try-catch structure and no further catch blocks are checked.

d) Explain why multithreading is preferred to a single thread in a client-server application? (2 marks)

Multithreading is preferred in a client-server application because it allows the server to handle

multiple client requests simultaneously, improving the overall responsiveness and efficiency of the server. Each client can be served in a separate thread, which prevents one client's request from blocking others and allows better utilization of system resources.

e) Name the two (2) classes that allow multithreading programming in Java. (2 marks)

1. `Thread` class
2. `Runnable` interface

f) Every Java thread has a feature that helps the operating system determine the order in which threads are scheduled. Explain how you can change this scheduling order. (5 marks)

In Java, thread priority helps the operating system determine the order in which threads are scheduled. Each thread has a priority, and higher-priority threads are usually executed in preference to lower-priority threads. You can change the scheduling order by setting the priority of a thread using the `setPriority` method. The priority is an integer value between `Thread.MIN_PRIORITY` (1) and `Thread.MAX_PRIORITY` (10), with `Thread.NORM_PRIORITY` (5) as the default.

Example:

```
Thread thread1 = new Thread(() -> {
    // thread code
});
thread1.setPriority(Thread.MAX_PRIORITY);

Thread thread2 = new Thread(() -> {
    // thread code
});
thread2.setPriority(Thread.MIN_PRIORITY);

thread1.start();
thread2.start();
```

g) Multi-threaded programs may often come to a situation where multiple threads try to access the same resources and finally produce erroneous and unforeseen results. Explain how this can be avoided in Java. (5 marks)

This situation is known as a race condition and can be avoided by synchronizing the access to

shared resources. Java provides several mechanisms to achieve synchronization:

1. Synchronized Methods:
   You can declare a method as synchronized, ensuring that only one thread can execute the method at a time for a given object.

```
public synchronized void synchronizedMethod() {
    // synchronized code
}
```

2. Synchronized Blocks:
   You can synchronize a block of code within a method. This allows for finer control over synchronization.

```
public void someMethod() {
    synchronized(this) {
        // synchronized code
    }
}
```

3. Locks:
   Java provides explicit lock objects in the `java.util.concurrent.locks` package, such as `ReentrantLock`, which offer more sophisticated thread synchronization mechanisms.

```java
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class SharedResource {
    private Lock lock = new ReentrantLock();

    public void accessResource() {
        lock.lock();
        try {
            // access shared resource
        } finally {
            lock.unlock();
        }
    }
}
```

4. **Atomic Variables:**
   For simple atomic operations, Java provides classes in the `java.util.concurrent.atomic` package, such as `AtomicInteger`, which can be used to perform thread-safe operations without explicit synchronization.

```java
import java.util.concurrent.atomic.AtomicInteger;

public class Counter {
    private AtomicInteger count = new AtomicInteger(0);

    public void increment() {
        count.incrementAndGet();
    }

    public int getCount() {
        return count.get();
    }
}
```

By using these synchronization mechanisms, you can prevent multiple threads from concurrently modifying shared resources, thereby avoiding race conditions and ensuring the correctness of your program.