Q1. Demonstrate method overloading using a Calculator class.

```
public class Calculator {
int add(int a, int b) {
return a + b;
}
double add(double a, double b) {
return a + b;
}
public static void main(String[] args) {
Calculator calc = new Calculator();
System.out.println("Sum (int): " + calc.add(10, 20));
System.out.println("Sum (double): " + calc.add(10.5, 20.5));
}
}
```

Q2. Demonstrate constructor overloading using a Book class.

```
public class Book {
String title;
int pages;
Book() {
title = "Unknown";
pages = 0;
}
Book(String title) {
this.title = title;
this.pages = 100;
}
Book(String title, int pages) {
this.title = title;
this.pages = pages;
}
void display() {
System.out.println("Title: " + title + ", Pages: " + pages);
}
public static void main(String[] args) {
Book b1 = new Book();
Book b2 = new Book("Java Basics");
Book b3 = new Book("Advanced Java", 500);
b1.display();
b2.display();
b3.display();
}
}
```

Q3. Create an abstract class Shape with subclasses Circle, Rectangle and override abstract methods.

```
abstract class Shape {
abstract void draw();
}
class Circle extends Shape {
void draw() {
System.out.println("Drawing Circle");
}
}
class Rectangle extends Shape {
void draw() {
System.out.println("Drawing Rectangle");
}
public static void main(String[] args) {
Shape s1 = new Circle();
Shape s2 = new Rectangle();
s1.draw();
s2.draw();
}
}
```

Q4. Implement an interface Printable with classes Document, Image.

```
interface Printable {
void print();
}
class Document implements Printable {
public void print() {
System.out.println("Printing Document");
}
}
class Image implements Printable {
public void print() {
System.out.println("Printing Image");
}
public static void main(String[] args) {
Printable p1 = new Document();
Printable p2 = new Image();
p1.print();
p2.print();
}
}
```

Q5. Create a user-defined exception class and apply it.

```
class InvalidAgeException extends Exception {
InvalidAgeException(String s) {
super(s);
}
}
public class CustomExceptionDemo {
static void validate(int age) throws InvalidAgeException {
if (age < 18)
throw new InvalidAgeException("Age is less than 18");
else
System.out.println("Valid age");
}
public static void main(String[] args) {
try {
validate(16);
} catch (InvalidAgeException e) {
System.out.println("Caught: " + e.getMessage());
}
}
}
```

Q6. Create a simple login form using JFrame, JLabel, JTextField, JButton.

```
import javax.swing.*;
public class LoginForm {
public static void main(String[] args) {
JFrame frame = new JFrame("Login Form");
JLabel userLabel = new JLabel("Username:");
JTextField userText = new JTextField();
JLabel passLabel = new JLabel("Password:");
JPasswordField passText = new JPasswordField();
JButton loginButton = new JButton("Login");
frame.setSize(300, 200);
frame.setLayout(null);
userLabel.setBounds(20, 20, 80, 25);
userText.setBounds(100, 20, 165, 25);
passLabel.setBounds(20, 60, 80, 25);
passText.setBounds(100, 60, 165, 25);
loginButton.setBounds(100, 100, 80, 25);
frame.add(userLabel);
frame.add(userText);
frame.add(passLabel);
frame.add(passText);
frame.add(loginButton);
frame.setVisible(true);
}
}
```

Q7. Implement single and multilevel inheritance using real-world classes like Person, Employee, Manager.

```
class Person {
String name;
void showPerson() {
System.out.println("Person Name: " + name);
}
}
class Employee extends Person {
int empId;
void showEmployee() {
System.out.println("Employee ID: " + empId);
}
}
class Manager extends Employee {
String department;
void showManager() {
System.out.println("Department: " + department);
}
public static void main(String[] args) {
Manager m = new Manager();
m.name = "Alice";
m.empId = 101;
m.department = "HR";
m.showPerson();
m.showEmployee();
m.showManager();
}
}
```

Q8. Use layout managers: FlowLayout, BorderLayout, GridLayout with components.

```
import javax.swing.*;
import java.awt.*;
public class LayoutExample {
public static void main(String[] args) {
JFrame frame = new JFrame("FlowLayout Example");
frame.setLayout(new FlowLayout());
frame.add(new JButton("Button 1"));
frame.add(new JButton("Button 2"));
frame.setSize(300, 100);
frame.setVisible(true);
JFrame borderFrame = new JFrame("BorderLayout Example");
borderFrame.setLayout(new BorderLayout());
borderFrame.add(new JButton("North"), BorderLayout.NORTH);
borderFrame.add(new JButton("South"), BorderLayout.SOUTH);
borderFrame.add(new JButton("East"), BorderLayout.EAST);
borderFrame.add(new JButton("West"), BorderLayout.WEST);
borderFrame.add(new JButton("Center"), BorderLayout.CENTER);
borderFrame.setSize(300, 200);
borderFrame.setVisible(true);
JFrame gridFrame = new JFrame("GridLayout Example");
    gridFrame.setLayout(new GridLayout(2, 3)); // 2 rows, 3 columns
    gridFrame.add(new JButton("1"));
    gridFrame.add(new JButton("2"));
    gridFrame.add(new JButton("3"));
    gridFrame.add(new JButton("4"));
    gridFrame.add(new JButton("5"));
    gridFrame.add(new JButton("6"));
```

```
        gridFrame.setSize(300, 150);
        gridFrame.setVisible(true);
    }
    }
```

Q9. Implement default, parameterized, and copy constructors in a class.

```
public class Student {
String name;
int age;
// Default constructor
Student() {
name = "Default";
age = 0;
}
// Parameterized constructor
Student(String name, int age) {
this.name = name;
this.age = age;
}
// Copy constructor
Student(Student s) {
this.name = s.name;
this.age = s.age;
}
void display() {
System.out.println("Name: " + name + ", Age: " + age);
}
public static void main(String[] args) {
Student s1 = new Student();
Student s2 = new Student("Alice", 22);
Student s3 = new Student(s2);
s1.display();
s2.display();
s3.display();
```

```
}
}
```

Q1. Write a program to implement bubble sort.

```
#include <iostream>
using namespace std;

int main() {
   int arr[100], n, i, j, temp;

   cout << "Enter the number of elements: ";
   cin >> n;

   cout << "Enter " << n << " elements:\n";
   for(i = 0; i < n; i++) {
      cin >> arr[i];
   }

   // Bubble Sort Logic
   for(i = 0; i < n - 1; i++) {
      for(j = 0; j < n - i - 1; j++) {
         if(arr[j] > arr[j + 1]) {
            // Swap elements
            temp = arr[j];
            arr[j] = arr[j + 1];
            arr[j + 1] = temp;
         }
      }
   }

   cout << "\nArray after Bubble Sort (Ascending Order): ";
   for(i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;

   return 0;
}
```

Q2. Write a program to implement insertion sort.

```
#include <iostream>
using namespace std;

int main() {
   int arr[100], n, i, j, key;

   cout << "Enter the number of elements: ";
   cin >> n;

   cout << "Enter " << n << " elements:\n";
   for(i = 0; i < n; i++) {
      cin >> arr[i];
   }

   // Insertion Sort Logic
   for(i = 1; i < n; i++) {
      key = arr[i];
      j = i - 1;

      // Move elements greater than key one position ahead
      while(j >= 0 && arr[j] > key) {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }

   cout << "\nArray after Insertion Sort (Ascending Order): ";
   for(i = 0; i < n; i++) {
      cout << arr[i] << " ";
   }
   cout << endl;

   return 0;
}
```

Q3. Write a program to search the elements in the linked list and display the same.

```
#include <iostream>
using namespace std;
// Define a node structure
struct Node {
   int data;
   Node* next;
};
Node* head = NULL;
// Function to create a linked list
void createList(int n) {
   Node *newNode, *temp;
   int data;
   if(n <= 0) {
      cout << "Invalid number of nodes!\n";
      return;
   }
   cout << "Enter data for node 1: ";
   cin >> data;
   head = new Node();
   head->data = data;
   head->next = NULL;
   temp = head;
   for(int i = 2; i <= n; i++) {
      newNode = new Node();
      cout << "Enter data for node " << i << ": ";
      cin >> data;
      newNode->data = data;
      newNode->next = NULL;
      temp->next = newNode;
      temp = temp->next;
   }
   cout << "\nLinked List Created Successfully!\n";
}
// Function to display he linked list
void displayList() {
   Node* temp = head;
   if(head == NULL) {
      cout << "List is empty.\n";
      return;
   }
   cout << "Linked List Elements: ";
   while(temp != NULL) {
      cout << temp->data << " ";
      temp = temp->next;
   }
   cout << endl;
}
// Function to search for an element in the list
void searchElement(int key) {
```

```
    Node* temp = head;
    int position = 1;
    bool found = false;
    if(head == NULL) {
        cout << "List is empty.\n";
        return;
    }
    while(temp != NULL) {
        if(temp->data == key) {
            cout << "Element " << key << " found at position " << position << ".\n";
            found = true;
            break;
        }
        temp = temp->next;
        position++;
    }
    if(!found)
        cout << "Element " << key << " not found in the list.\n";
}
// Main program (menu-driven)
int main() {
    int n, choice, key;
    do {
        cout << "\n=== MENU ===\n";
        cout << "1. Create Linked List\n";
        cout << "2. Display Linked List\n";
        cout << "3. Search an Element\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch(choice) {
            case 1:
                cout << "Enter number of nodes: ";
                cin >> n;
                createList(n);
                break;
            case 2:
                displayList();
                break;
            case 3:
                cout << "Enter the element to search: ";
                cin >> key;
                searchElement(key);
                break;
            case 4:
                cout << "Exiting program...\n";
                break;
            default:
                cout << "Invalid choice! Try again.\n";
        }
```

```
    } while(choice != 4);
    return 0;
}
```

Q4. Write a program to implement the concept of Stack with Push, Pop, Display and Exit operations.

```
#include <iostream>
using namespace std;

#define MAX 100   // Maximum size of stack

class Stack {
   int arr[MAX];
   int top;

public:
   Stack() {
      top = -1;
   }

   // Function to push element onto stack
   void push(int value) {
      if(top == MAX - 1) {
         cout << "Stack Overflow! Cannot push " << value << endl;
      } else {
         top++;
         arr[top] = value;
         cout << value << " pushed onto the stack.\n";
      }
   }

   // Function to pop element from stack
   void pop() {
      if(top == -1) {
         cout << "Stack Underflow! No elements to pop.\n";
      } else {
         cout << arr[top] << " popped from the stack.\n";
         top--;
      }
   }

   // Function to display stack elements
   void display() {
      if(top == -1) {
         cout << "Stack is empty.\n";
      } else {
         cout << "Stack elements (Top to Bottom): ";
         for(int i = top; i >= 0; i--) {
            cout << arr[i] << " ";
         }
         cout << endl;
      }
   }
```

```
};

// Main Function
int main() {
    Stack s;
    int choice, value;

    do {
        cout << "\n=== STACK MENU ===\n";
        cout << "1. Push\n";
        cout << "2. Pop\n";
        cout << "3. Display\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;

        switch(choice) {
            case 1:
                cout << "Enter value to push: ";
                cin >> value;
                s.push(value);
                break;

            case 2:
                s.pop();
                break;

            case 3:
                s.display();
                break;

            case 4:
                cout << "Exiting program...\n";
                break;

            default:
                cout << "Invalid choice! Try again.\n";
        }

    } while(choice != 4);

    return 0;
}
```

Q5. Write a program to search the element using binary search.

```
#include <iostream>
using namespace std;

int main() {
   int arr[100], n, i, key;
   int low, high, mid;
   bool found = false;

   cout << "Enter the number of elements: ";
   cin >> n;

   cout << "Enter " << n << " elements in **sorted order** (ascending):\n";
   for(i = 0; i < n; i++) {
      cin >> arr[i];
   }

   cout << "Enter the element to search: ";
   cin >> key;

   // Binary Search Logic
   low = 0;
   high = n - 1;

   while(low <= high) {
      mid = (low + high) / 2;

      if(arr[mid] == key) {
         cout << "\nElement found at position " << mid + 1 << endl;
         found = true;
         break;
      }
      else if(arr[mid] < key)
         low = mid + 1;
      else
         high = mid - 1;
   }

   if(!found)
      cout << "\nElement not found in the array.\n";

   return 0;
}
```