Q1. Explain the merits and demerits of oops

➔ OOP uses concepts such as Objects, Classes, Inheritance, polymorphism, data encapsulation and abstraction

➔ Merits:

- ➔ Reuse the code multiple times using class
- ➔ Inherit class to subclass for data redundancy
- ➔ It is easy to maintain and modify
- ➔ Maintains security of data
- ➔ Low cost development

➔ Demerits

- ➔ Size is larger than other programs
- ➔ Requires a lot of effort to create
- ➔ It is slower than other programs
- ➔ Not suitable for some sorts of problems
- ➔ It takes time to get used to it

Q2. Differentiate between object-oriented programming and procedural oriented programming language

Object oriented programming	Precedural oriented programming
Program is divided into small parts called objects	Program is divided into small parts called functions
Follows bottom up approach	Follows top down approach
Has access specifiers like private public protected	There is no access specifier
Adding new data and function is easy	Adding new data and function is not easy
Eg. C++ Java, python, C# , etc	Eg. C, fortran, Pascal, Basic, etc

Q3. Write a program to display pattern

```
public class pattern {
    public static void main(String[] args){
        //for loop to go from 5 to 1
        //Reduces number of numbers in each line
        for(int i=5; i>0; i--){
            //for loop to go from 1 to 5
            //prints the numbers in sequence
            for(int j=1; j<= i; j++){
                System.out.print(j + " ");
            }
            //goes to next line
            System.out.println();
        }
    }
}
```

Q4. List 5 basic datatypes used in java with example

```
public class datatypes {
    public static void main(String[] args){
        boolean isTrue = true;
        int num1 = 35;
        float num2 = 34.6f;
        long longnum = 100000l;
        char letter = "j";
    }
}
```

Q5. Elaborate inheritance and its types. Explain any 1 type with appropriate example

➔ There are 3 types of inheritance:

- ➔ Single inheritance
  - ➔ A structure having one and only one parent as well as child class
  - ➔ Child class is authorized to access the property of Parent class
- ➔ Multilevel inheritance
  - ➔ Standard structure of single inheritance having one Parent, one or more intermediate and one child class
  - ➔ Child class as well as intermediate class may access the properties of upper level classes
- ➔ Hierarchical inheritance
  - ➔ A structure having one parent and more child class
  - ➔ Child classes mush be connected with only parent class

➔ Example: single inheritance

```
class Employee{
    float salary = 40000;
}
public class single_inheritance extends Employee{
    int bonus = 10000;
    public static void main(String[] args){
        single_inheritance sp = new single_inheritance();
        System.out.println("Pprogramminig salary is: "+ sp.salary);
        System.out.println("Bonus is: "+ sp.bonus);
    }
}
```

Q6. Briefly explain about basic concepts of object-oriented programming language

➔ Basic concepts of object oriented programming

- ➔ Object
  - ➔ Any entity that has state and behaviour is known as an object
  - ➔ Can be defined as an instance
- ➔ Class
  - ➔ The class is a group of similar entities
  - ➔ A class can also be defined as a blueprint from which you can create an individual object
- ➔ Inheritance
  - ➔ When one object acquires all the properties and behaviours of a parent object it is know as inheritance
- ➔ Polmorphism
  - ➔ If one task is parformed in different ways, it is known as polymorphism
- ➔ Abstraction
  - ➔ Hiding internal details and showing functionality is known as abstraction
- ➔ Encapsulation
  - ➔ Binding code and data together into a single unit are known as encapsulation

Q7. Display factors

```
public class Factors{
    public static void main(String[] args){
        //input number
        int num = 60;
        //prints output
        System.out.print("Factors are: ");
        //for loop to print out all factors
        for(int i = 1; i ⩽ num; i++){
            //if statement to find out the factors and print it
            if(num % i == 0){
                System.out.print(i + " ");
            }}}}
```

Q8. Write about various decision-making statement available in java (if, if-else, nested if-else, switch) explain one with example

➔ If Statement

- ➔ If the condition is true, the statement is executed. If it is false, the statement is skipped
- ➔ `if(condition){`
- ➔ `statement;`
- ➔ `}`

➔ if-else condition

- ➔ An else clause can be added to an if statement to make an if else statement
- ➔ `if(condition){`
- ➔ `statement;`
- ➔ `} else {`
- ➔ `statement;`
- ➔ `}`

➔ nested if else

- ➔ The statement is executed as a result of an if statement or else clause could be another if statement
- ➔ `if(condition){`
  - ➔ `if(condition){`
    - ➔ `statement1;`
    - ➔ `}`
  - ➔ `statement2;`
  - ➔ `}else{`
  - ➔ `statement3;`
- ➔ `}`

➔ switch statement

- ➔ Switch statement evaluates an expression, then attempts to match the result to one of several possible cases
- ➔ `switch(expression){`
  - ➔ `case value1:`
    - ➔ `statement-list1`
  - ➔ `case value2:`

➔ statement-list2

➔ }

Unit2

Q1. program to display reverse

```
public class reverse {
    public static void main (String[] args){
        Scanner sc = new Scanner(System.in);
        //Askes for number input
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        //Declares vairable for reverse of input
        int rev = 0;
        //example number will be 25
        //while loop to reverse number
        while(num≠0){
            //takes remainder when number is divided by 10
            int rem = num % 10;
            //rem will be 5
            //reverse is assigned the number of initial reverse multiplied by 10 + remainder
            rev = rev * 10 + rem;
            //rev will be 5
            num = num/10;
            //num will be set to 2
        }
        System.out.println("Reverse is "+ rev);
    }
}
```

Q2. Swing code for following application

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

public class swing_code {
    public static void main(String[] args){
        //initiate frame
        JFrame f = new JFrame();
        //making elements
        JLabel l1 = new JLabel("Application for reverse");
        JLabel l2 = new JLabel("Enter any number");
        JTextField tf1 = new JTextField();
        JLabel l3 = new JLabel("Answer");
        JTextField tf2 = new JTextField();
        JButton b1 = new Jbutton("Check");
        //adding event listener to button
        b1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                //calculating if its a palindrome
                int num1 = Integer.parseInt(tf1.getText());
                int num2 = Integer.parseInt(tf2.getText());
                int rev = 0;
                while(num1≠0){
                    int rem = num1 % 10;
                    rev = rev*10 +rem;
                    num1=num1/10;
                }
                tf2.setText(Integer.toString(rev));
                if(num1 == num2){
                    System.out.println("It is a palindrome");
                }else{
```

```
                    System.out.println("it is not a paplindrome");
                }
            }});

        JButton b2 = new Jbutton("Cancel");
        b2.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                tf1.setText("");
                tf2.setText("");
            }});
        //adding elements to frame
        f.add(l1);
        f.add(l2);
        f.add(tf1);
        f.add(l3);
        f.add(tf2);
        f.add(b1);
        f.add(b2);
    }
}
```

Q3. Write code to generate multiplication table

```
public class multable {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        //take input from user
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        //for loop to print multiplication table
        for(int i=1; i<=10; i++){
            System.out.println(num + " * " + i + " = " + num*i);
        }
    }
}
```

Q4. Write swing code to create application for student

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

public class student {
    public static void main(String[] args){
        JFrame f = new JFrame();

        JLabel l = new JLabel("Student application");
        JLabel l1 = new JLabel("First Name");
        JTextField tf1 = new JTextField();
        JLabel l2 = new JLabel("Last Name");
        JTextField tf2 = new JTextField();
        JLabel l3 = new JLabel("Address");
        JTextField tf3 = new JTextField();
        JLabel l4 = new JLabel("email");
        JTextField tf4 = new JTextField();
        JLabel l5 = new JLabel("id");
        JTextField tf5 = new JTextField();
        JButton b1 = new JButton("Submit");
        b1.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                if(tf1.getText().length() > 0 &&
tf2.getText().length() > 0){
                    JOptionPane.showMessageDialog(b1, "Logged
in");
                }else{
                    JOptionPane.showMessageDialog(b1, "Invalid id
password");
                }
            }
```

```
        });
        JButton b2 = new JButton("Clear");
        b2.addActionListener(new ActionListener(){
            public void actionPerformed(ActionEvent e){
                tf1.setText("");
                tf2.setText("");
            }
        });
    }
}
```

Q5. Palindrome

```
public class palindrome {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        //Askes for number input
        System.out.print("Enter a number: ");
        int num = sc.nextInt();
        //temp to check if the number is a palindrome later
        int temp = num;
        //Declares vairable for reverse of input
        int rev = 0;
        //example number will be 25
        //while loop to reverse number
        while(num≠0){
            //takes remainder when number is divided by 10
            int rem = num % 10;
            //rem will be 5
            //reverse is assigned the number of initial reverse
multiplied by 10 + remainder
            rev = rev * 10 + rem;
            //rev will be 5
```

```
            num = num/10;
            //num will be set to 2
        }
        if(temp == rev){
            System.out.println("Number is a palindrome");
        }else{
            System.out.println("Number is not a palindrome");
        }
    }
}
```

Q6. Overriding with example

➔ Overriding is a feature exclusive to object oriented programming

➔ It allows subclass or child class to provide a specific implementation of a method that is already provided by one of it super-classes or parent classes

➔ Overriding is a way in which java achieves runtime polymorphism

➔ If a class is used to invoke the method then the parent class will be executed but if an object of the subclass is used to invoke the method the child class will be executed

➔ Example

```
class Parent{
    void show(){
        System.out.println("Parent shown");
    }
}
class Child extends Parent{
    void show(){
        System.out.println("Child shown");
    }
}
public class overriding {
    public static void main(String[] args){
        Parent p1 = new Parent();
        p1.show();
```

```
        Parent p2 = new Child();
        p2.show();
    }
}
```

Q7.What is class. Explain how to access class member function with examples

➔ A class is a user defined blueprint or prototype from which objects are created

➔ Class declaration can use the following components

- ➔ Modifiers: public, private, protected
- ➔ Class name: Name should begin with captal letter
- ➔ Superclass (if any)
- ➔ Interfaces(if any)
- ➔ Body

➔ Example

```
class Base{
    void print(){
        System.out.println("Class printed");
    }
}
public class example_class extends Base{
    public static void main(String[] args){
        Base b = new Base();
        b.print();
    }
}
```

Q8. Method overloading

➔ method overloading allows different methods to have the same name but different signatures

➔ there signatures can differe by number of input parametets or type of input parameters or both

➔ It is form of compile time polymorphism

```
public class Unit2_Q8 {
    //method to calculate area of square
    void calarea(int s){
        int area = s^2;
        System.out.println("Area of Square is "+ area);
    }
    //method to calculate area of triangle
    void calarea(int b, int h){
        int area = (b*h)/2;
        System.out.println("Area of Triangle is "+ area);
    }
    //main method
    public static void main(String[] args){
        //calling class as an object
        Unit2_Q8 Area =  new Unit2_Q8();
        //calling first method (area of square)
        Area.calarea(5);
        //calling second method (area of triangle)
        Area.calarea(5, 6);
    }
}
```

Q9. What is interface? Explain forms of implementing interfaces with examples

➔ Interface is a blueprint of a class.

➔ Java interface contains static and abstract methods

➔ This is a mechanism to achieve abstraction

➔ It specifies what a class must do, but does not tell how

➔ If class implements interface and does no provide method bodies then the class is declared abstract

```
//defining first interface
interface base1 {
    default void base1(){
```

```
        System.out.println("Base1 interface printed");
    }
}
//defining second interface
interface base2 {
    default void base2(){
        System.out.println("Base2 interface printed");
    }
}
//main class with both interfaces implemented
public class Unit2_Q9 implements base1, base2{
    //method involving both interfaces
    public void imp(){
        //calling constructors from both interfaces
        base1.super.base1();
        base2.super.base2();
    }
    //main method
    public static void main(String[] args){
        //calling main class as object i
        Unit2_Q9 i = new Unit2_Q9();
        //calling method from the class
        i.imp();
    }
}
```

Q10.

Q2. Explain steps for JDBC connection

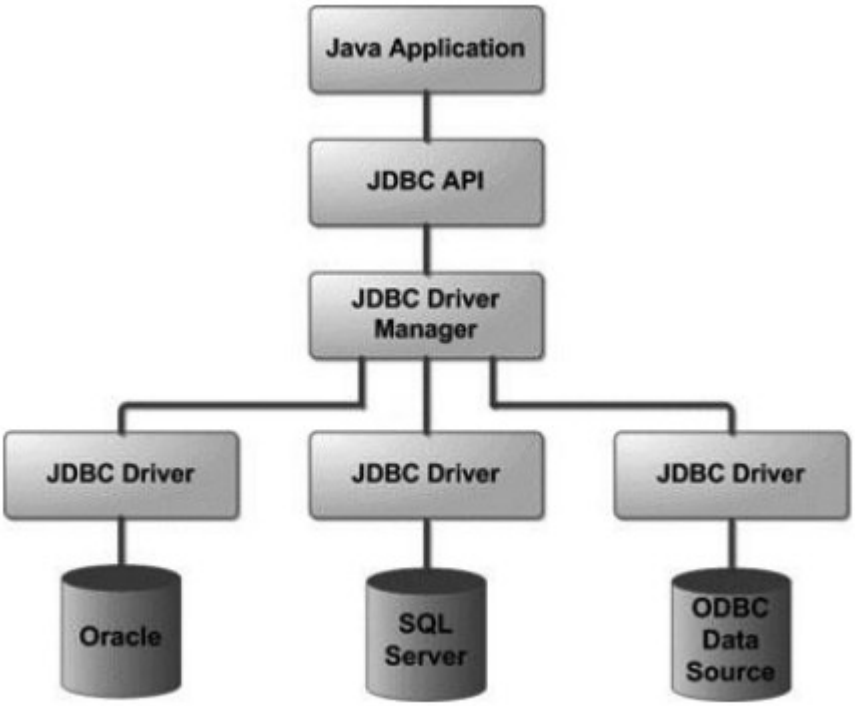
➔ Steps for JDBC connection are:

- Import JDBC Driver
  - Import statements tell the Java compiler where to find the classes to refer in the code
  - To use the standard JDBC driver we import it as
    - Import java.sql.*;
- Register JDBC Driver
  - We must register the driver in our program before using it
  - Registering the driver can be done by two approaches
    - Approach 1
      - Class.forName("com.mysql.cj.jdbc.Driver")
    - Approach 2
      - Driver myDriver = new oracle.jdbc.driver.OracleDriver();
      - DriverManager.registerDriver(myDriver);
- Database URL formalation
-

RDBMS	JDBC driver name	URL format
MySQL	com.mysql.jdbc.Driver	jdbc:mysql://hostname/ databaseName
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2:hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds:hostname: port Number/databaseName

- Create Connection object
  - Connection object is created in the format
    - Database url
    - Username
    - Password
  - Connection con = DriverManager.getConnection(URL, USER, PASS)

Q2.



Q5. Explain different types of JDBC drivers

➔ There are 4 types of JDBC drivers

- JDBC-ODBC bridge driver
  - This is integrated in Java.
  - Suitable to run with a local database.
  - Needs the client to install database software.
  - Performance is slow
- Native API driver
  - Needs configuration before installing native code on client side
  - Dependent of DBMS.
  - Client can connect directly to database server
  - Provides adequate speed to access database.
- Network Protocol Driver
  - This is the most flexible configuration
  - Needs an intermediate server to function which has to be configured by the vendor
  - Can access multiple databases from one driver
  - Independent of DBMS
- Thin Driver
  - This is the fastest way to communicate SQL queries to the DBMS
  - This allows direct call to database without client pre-configuration
  - A different driver needs to be loaded foreach DBMS
  - Not appropriate for Applets

Q6. Diferrence between type 1 and type 2

➔ Though both type 1 and type 2 drivers are not written in Java, there was some significant difference between them. Type 2 driver has better performance than type 1 driver because of less layer of communication and translation. As opposed to the type 1 JDBC driver, in which JDBC calls are translated into ODBC calls before they go to the database, type 2 JDBC driver directly connects to DB client using the native library.

Q7. Difference between type 3 and type 4

➔ The main difference between type 3 and type 4 JDBC drivers was the removal of 3 tier architecture. Type 4 JDBC drivers directly connect to the database using their native protocol as opposed to the network protocol used by type 3 drivers. Though both type 3 and type 4 driver is written in Java.

➔ Another key difference is the ease of use, type 4 drivers just require one JAR file into classpath in order to connect to DB. The performance of the Type 4 JDBC driver is also better than the type 3 driver because of direct connectivity to the database as opposed to 3 tier architecture of the type 3 driver.

Q8.