Lab-1

In Java, a class can extend only one abstract class but can implement many interfaces; so multiple inheritance of type is acheved with interfaces not abstract classes.

Prefer an abstract class when:

1. we want to share common state. and some default method implementation.
2. classes have a clear "is-a relationship and are closely related.

Prefer and interface when:

1. We want to define a contract without shared state.
2. We want a class to play multiple roles eg: Comparable, Serializable, Runnable. So it can implement several.

Lab 2:

Encapsulation hides data inside a class using private fields and allows only through controlled public methods which validate input and prevent illigal changes.

Example:

```
class BankAccount {
Private string accountNumber;
private double balance;

Public void setAccountNumber(String Acc)
{
if (Acc no == null || acc No.trim() is
Empty()).
    throw new IlligalArgument Exception("
      Invalid Account Number") ;
      this.Account Number = );
```

```
        this.accountNumber = accNo;
    }

Public void setInitialBalance(double amount)
{
    if (amount < 0)
        throw new IllegalArgumentException("
            Balance cannot be negative");
        this.balance = amount;
    }
Public double getBalance();{
        return balance;
    }
}
```

Here accNo and amount cannot be changed directly from outside. So all updates go through validation logic preserving data integrity

Lab 3:

classes and Responsibilities.

=> RegistarParking → Parking request

=> ParkingPool → Shared synchronized queue.

=> ParkingAgent → Worker thread.

=> Mainclass → Simulates car arrival.

Code:

```
import java utis *;
class RegistarParking {
String carNo;
RegistarParking (String carNo){
This.carNo = carNo;
System.out.println("car"+ can not
"requested Parking");
} }
```

```
class ParkingPool {
Queue<RegistarParking> queue = new
linkedList<>();
Synchronized void addCar(Register
Parking car){
 queue.add(car);
  notify();
 }
Synchronized RegistarParking.Park
car() throws InterruptedException
{
While (queue is Empty() wait();
 return queue poll();
 }
}
class ParkingAgent extends Thread{
 ParkingPool pool string.name){
```

```
Public void Main() {
try {
Registor Parking Car = Pool.Park Car();
.System.out.println(agent.name+"
parked Car>4 Car.Car No + "");
}.catch (Exception e){} ?
}
}
public class Mainclass {
public static void main (String [] args){
new parking Agent (Pool, "Agent 1").
stantic);
new parking Agent (Pool, "Agent 2")
start (),
pool.add Car (new Registor Parking) ("ABC")
pool.add car (new RegistorParking)("
XYZ456"));
}}
```

Lab 4:

How JDBC Works.

Java App → JDBC API → Driver → Database

→ Result.

Steps:

1. Load driver.
2. Get connection.
3. Create statement.
4. Execute query.
5. Process Resultset.
6. Close Resources.

Code:

```
Connection con = null.
Statement st = null.
Resultset rs = null.

try {
    con = DriverManager.getConnecti
```

```
MySql:"Localhost+(db"."root","");
'1t =con.@Dreatstatement();
rs .= st.executeQuery("select *
    from student");

while(MS.Next()){
System.out.println(rs.getInt("id")
+"  "+ rs getString("Name"));
}
}catch(SQLExceptione){
  e.printStackTrace();
}
 finally{
  try{if(rs!=null)rs.close();}
   catch(Exceptione){}
try{if(est!=null)stclose();}
```

```
finally { if (conn != null) conn.close(); }
    catch (Exception e) { }
}
```

Lab 5:

Controller Role:

1. Receives request:
2. Calls model.
3. Sends data to view (JSP).

Servlet:

```
@WebServlet("student")
Public class Student Servlet extends HHP
Servlet {
protected Void do get (HHP Servlet
Request req, HHP Servulet Response
res).
throws Servulet Exception, IP Exceptio
n {
  req.Set Attribute ("Name", "Samiul");
 Reques Dispatcher rd = req.getRe
quest Dispatcher ("student.jsp");
, rd.forword (red, res);
```

Lab-6:

Prepared Statement Precompiles the said Once and lets execute it multiple times with different Parameters. Which improves performance for repeated queries and avoids SQL from data values.

Code:

```
String sql = "Insert into student
(name, age)
    values (?, ?)";
prepared statement ps = con.pre
pare statement (SQL);
  ps. set string (1, "Rahim");
  ps set Int (2, 20);
  ps execute update ();
```

Lab 7.

Result set is a cursor like object that holds the rows returned by a SELECT statement; you move row by row. using next() and read column value s with getters like getString() and getInt().

next(): moves the cursor to the next row returns false when there is no more row.

getString(columnName or Index): reads a text value from the current row.

getint(columnName or index) reads an integer value from the current row,

```
Statement stant = Conn.Create state
ment();
Resultset rs = stmt.eucuteQuery
(SQL)) {
while (rs.next() {
  int id = rs.getInt("d")
  String name = rs.getString("Name
  intage = rs.gtInt("age");
System.out.println(id+"_"+name
"_"+age.);
}
Catach (SQLExeception e){
  e.printStackTrace();
}
```