Name: Suryaprabha K

Domain: BFS

Assignment:26-03-2022

1.How will you achieve data encapsulation?

* Encapsulation is achieved by assigning the members of the class as private.
* To access the private members, getter and setter methods are used.

2.What is static?

* Static is a keyword that belongs to class and not to objects.
* Static members and methods can be accessed without object creation.
* Static members can be accessed by using the class name.
* Memory allocation is done only once at the time of declaration of static members.
* Static methods cannot be inherited from their subclasses because they belong to the class in which they are declared.
* Similarly, we cannot use a static variable in its subclasses.

3.Why static?

* Static is used for memory management.
* To manage memory efficiently, static is used.
* To store data which is common to all the objects, static is used since the memory allocated by the static members is accessed /shared by all the objects.
* Example:

To store a college name which is common to all students, to store a company name which is common to all the employees, static keyword is used while declaring.

4.Access specifiers:

* Private:
* Private access modifier is used when we need to access the members and methods only within a class.
* Private members cannot be accessed outside the class, within a package, outside the package.
* Private access modifier is used when there is a need to store confidential data or to protect the data from unauthorized users.
* Protected:
* Protected access modifier is used when we need to access the members and methods within the class, outside the class, within the package, outside the package only by using subclasses.
* Protected members and methods cannot be accessed outside the package directly.
* Public:
* Public access modifier is used when we need to access the members and methods everywhere.
* It can be accessed within the class, outside the class, within the package, outside the package only through child-class, outside the package.
* Default:
* When we don’t declare any access specifier explicitly in program, the compiler automatically consider those members and methods as default.
* The access level of default modifiers is within the class, outside the class, within the package.
* It cannot be accessed outside the package through sub class and outside the package directly.

5.Nonaccess specifiers:

* Static:
* Static is a keyword that belongs to class and not to objects.
* Static members and methods can be accessed without object creation.
* Static members can be accessed by using the class name.
* Memory allocation is done only once at the time of declaration of static members.
* Final:
* Final keyword is used where we need to store constant values or where we don’t need to change the value of final members, methods or classes.
* Once initialized, the final variables, methods, classes cannot be changed.
* If we try to change, compiler error occurs.
* Once a class is declared as final, it cannot be inherited.
* Final keyword is used when we need to restrict the inheritance property for classes.
* Abstract:
* Abstract keyword is used in classes, methods to achieve abstraction.
* Abstract keyword is not used for variables.
* Abstract keyword is used when a method has no implementation and only declaration.
* The implementation should be done in child classes.
* When class consists of at least 1 abstract method, to restrict the access of abstract methods by using objects, the class should be declared as abstract.
* Abstract methods cannot be synchronized, static, final, native, private.
* Synchronized:
* Transient:
* Volatile:

6.Static and Instance:

|  |  |
| --- | --- |
| STATIC | INSTANCE |
| Static members, methods belong to class and not object. | Instance members, methods belong to object and not class. |
| Static members, methods can be accessed without object creation. They are accessed with the help of class name. | Instance members, methods can be accessed with only object creation. |
| Static methods can access only static members. | Instance methods can access both static and instance members. |
| Memory allocation is done only once for static members and it is shared by all the objects. | For each instance we are creation, each copy of instance variable is created in the heap memory. |
| When we change the value of static members, it will be affected in all objects. | When we change the value of instance members, it will be not affected in other objects. |

7.Is static constructor possible in java? Justify.

* Static constructor is not possible in java.
* Static constructor cannot be overloaded.
* As static belongs to class and constructor belongs to object (i.e.), It is used to initialize the members of the object, static constructor is not possible in java.
* A constructor is called when an object of a class is created, so no use of the static constructor.
* Static methods cannot be inherited from their subclasses because they belong to the class in which they are declared. Similarly, we cannot use a static variable in its subclasses.
* So, it is illegal to declare a constructor as static.
* The program will not be compiled and throw a compile-time error.

8. Copy constructor:

* A copy constructor in a Java class is a constructor that creates an object using another object of the same Java class.
* That's helpful when we want to copy a complex object that has several fields, or when we want to make a deep copy of an existing object.
* For example, Student s1 = s2, where Student is the class.
* It returns the duplicate copy of existing object of class.
* If a field is declared as final, copy constructor can be used to change which is the advantage of the copy constructor.
* Copy constructor can be used to avoid clone() method.

9.Constructor chaining:

* Constructor chaining is the process of calling one constructor from another constructor with respect to current object.
* Constructor chaining can be done in two ways:
* It can be done using this() keyword for calling constructors in same class
* By using super() keyword to call constructor from the base class.

10.Why Constructor chaining?

* This process is used when we want to perform multiple tasks in a single constructor rather than creating a code for each task in a single constructor we create a separate constructor for each task and make their chain which makes the program more readable.
* Constructor chaining allows us to create a separate constructor for each task and make links or chains among them, so as to increase the readability of the code.
* Suppose, if we do not perform chaining among constructors and they require a specific parameter, then we will need to initialize that parameter twice in each constructor.
* Whenever there is a need to make changes in the parameter you will have to make changes inside each constructor.
* In Java, it is invalid and illegal to call the constructor directly by name.
* We have to use either of these two keywords this() and super() to call a constructor.

11.Functions of JVM:

JVM performs the following functions. They are as follows,

* It loads .class code which is the bytecode generated by the javac compiler into the class loader.
* It is composed of bytecode verifier that verifies the code whether it is composed of any illegal code fragments in it.
* It executes the bytecode to machine code.
* Automatic Garbage Collector is present in JVM to automatically delete the unused or unreferenced objects thereby efficiently managing the memory.
* JVM comes with JIT(Just-in-Time) compiler that converts Java source code into low-level machine language. Hence, it runs more faster as a regular application.
* It provides runtime environment to execute the code.

12.How JVM handles the exception:

* Inside a method, if any exception occurs an exception object is created which consists of name, description of exception and current state of the program where the exception has occurred.
* The generated exception object is handover to the JVM.
* JVM checks for exception handler to handle that exception.
* Through the runtime stack mechanism, It searches backward through the call stack until it finds a matching exception handler for that particular class of Exception object.
* The run-time stack system searches the call stack to find the method that contains a block of code that can handle the occurred exception. The block of the code is called an Exception handler.
* The run-time system starts searching from the method in which the exception occurred, proceeds through the call stack in the reverse order in which methods were called.
* If it finds an appropriate handler then it passes the occurred exception to it. Appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.
* If the caller of the method that created exception doesn’t contain the exception handling code,the JVM terminates the method and the stack entry is deleted.
* If it doesn’t find any appropriate handler, JVM hand over the exception object to the Default Exception Handler which just prints the information about the exception to the consoleand terminates the program abruptly.
* The Default Exception Handler prints the information in the order of,
* Name of exception
* Description of exception
* Stack Trace
* Example:

If exception occurred at main method in Main class,

Exception in thread main:java.lang.ArithmeticException:/ by zero

at Main.main(DefaultExceptionHandler)

package com.practice;

public class DefaultExceptionHandler {

public static void main(String[] args)

{

DefaultExceptionHandler a=new DefaultExceptionHandler();

a.doStuff();

}

public void doStuff()

{

doMoreStuff();

}

public void doMoreStuff()

{

System.out.println(10/0);

}

}

Console:

Exception in thread "main" java.lang.ArithmeticException: / by zero

at com.practice.DefaultExceptionHandler.doMoreStuff(DefaultExceptionHandler.java:15)

at com.practice.DefaultExceptionHandler.doStuff(DefaultExceptionHandler.java:11)

at com.practice.DefaultExceptionHandler.main(DefaultExceptionHandler.java:7)

13.Which inheritance is required to achieve method over riding:

* There must be an IS-A relationship (inheritance).
* Overriding is achieved only when there is inheritance especially IS-A relationship inheritance.
* Method overriding can be achieved in single inheritance, multilevel and hierarchical inheritance.

14.Throw and Throws:

|  |  |
| --- | --- |
| THROW | THROWS |
| The throw keyword is used inside a function. It is used when it is required to throw an Exception logically. | The throws keyword is used in the function signature. It is used when the function has some statements that can lead to exceptions. |
| The throw keyword is used to throw an exception explicitly. It can throw only one exception at a time. | The throws keyword can be used to declare multiple exceptions, separated by a comma. Whichever exception occurs, if matched with the declared ones, is thrown automatically then. |
| throw keyword cannot propagate checked exceptions. It is only used to propagate the unchecked Exceptions that are not checked using the throws keyword. | throws keyword is used to propagate the checked Exceptions only. |
| Syntax of throw keyword includes the instance of the Exception to be thrown. Syntax wise throw keyword is followed by the instance variable. | Syntax of throws keyword includes the class names of the Exceptions to be thrown. Syntax wise throws keyword is followed by exception class names. |

15.Hierarchy of catch Block:

catch(UserDefinedException exceptionObject)

{

}

Catch(PreDefinedException exceptionObject)

{

}

Catch(PreDefinedException exceptionObject)

{

}

Catch(MiscellanousException exceptionObject)

{

}

Catch(Exception exceptionObject)

{

}

* Base class Exception should be at the last catch block in order to improve the efficiency while executing the program.
* We should not specify the base class Exception before the specific child class Exception catch block as the base class Exception itself enough to handle all the exceptions.
* The best practice is to provide the generated or specific exception class catch block at the first to improve efficiency of the program.

15.Best practice to handle exceptions:

* Throw keyword is the best practice to implement exception handling.

16.Purpose of throws:

* To declare multiple exceptions, throws is used.
* To provide information about the exceptions to the programmer as well as to the caller of the method that throws the exceptions.
* When a method declares that it throws an exception, it is not required to handle the exception.

17.Is multiple inheritance possible in java?

* Multiple inheritance is not possible in java through class.
* It is possible by implementing interface and not by extending the interface.

18.Life cycle of thread:

In Java, a thread always exists in any one of the following states. These states are:

* New
* Active
* Blocked / Waiting
* Timed Waiting
* Terminated

19.Multithreading:

Multithreading is a model of program execution that allows for multiple threads to be created within a process, executing independently but concurrently sharing process resources.

Console Application

EmployeeDatas.java

package employee;

import java.sql.\*;

import java.text.ParseException;

import java.text.SimpleDateFormat;

//import java.util.ArrayList;

import java.util.Date;

//import java.util.Iterator;

//import java.util.ListIterator;

import java.util.Scanner;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

class MainEmployee{

private String employeeID;

private String employeeName;

private String employeeEmail;

private String employeeDOB;

private String employeeDOJ;

private String employeeMobileNumber;

//super keyword invokes the MainEmployee constructor

MainEmployee(String employeeID,String employeeName,String employeeEmail,String employeeDOB,String employeeDOJ,String employeeMobileNumber)

{

this.employeeID=employeeID;

this.employeeName=employeeName;

this.employeeEmail=employeeEmail;

this.employeeDOB=employeeDOB;

this.employeeDOJ=employeeDOJ;

this.employeeMobileNumber=employeeMobileNumber;

}

//In order to access the private fields,getter methods are used.

public String getEmployeeID()

{

return employeeID;

}

public String getEmployeeName()

{

return employeeName;

}

public String getEmployeeEmail()

{

return employeeEmail;

}

public String getEmployeeDOB()

{

return employeeDOB;

}

public String getEmployeeDOJ()

{

return employeeDOJ;

}

public String getEmployeeMobileNumber()

{

return employeeMobileNumber;

}

/\*public String toString()

{

return employeeID+" "+employeeName+" "+employeeEmail+" "+employeeDOB+" "+employeeDOJ;

}\*/

//To perform overriding

public String validateEmployeeID()

{

return employeeID;

}

public String validateEmployeeName()

{

return employeeName;

}

public String validateEmployeeEmail()

{

return employeeEmail;

}

public String validateEmployeeDOB()

{

return employeeDOB;

}

public String validateEmployeeDOJ()

{

return employeeDOJ;

}

public String validateEmployeeMobileNumber()

{

return employeeMobileNumber;

}

}

public class EmployeeDatas extends MainEmployee{//child class

//constructor definition

EmployeeDatas(String employeeID,String employeeName,String employeeEmail,String employeeDOB,String employeeDOJ,String employeeMobileNumber)

{

super(employeeID,employeeName,employeeEmail,employeeDOB,employeeDOJ,employeeMobileNumber);//It invoke parent class constructor

}

Scanner scannerString=new Scanner(System.in);

Scanner scanner=new Scanner(System.in);

public String validEmployeeID()//Overriding the validEmployeeID method

{

System.out.println("Enter the employee ID:");

String employeeID=scannerString.nextLine();

if(!(employeeID.length()==7 && employeeID.matches("ACE+[0-9]{4}")))

{

System.out.println("EmployeeID must start with 'ACE' followed by 4 digits.Do not include any special characters");

validEmployeeID();

}

return employeeID;

}

public String validEmployeeName()//Overriding the validEmployeeName method

{

System.out.println("Enter the employee name:");

String employeeName=scannerString.nextLine();

Pattern patternName = Pattern.compile("^[a-zA-Z]+$");

Matcher matcherName=patternName.matcher(employeeName);

boolean checkName = matcherName.find();

if(checkName==false)

{

System.out.println("Employee name should contain only alphabets.Do not include any special characters or numerics");

validEmployeeName();

}

return employeeName;

}

public String validEmployeeEmail() //Overriding the validEmployeeEmail method

{

System.out.println("Enter the employee email:");

String employeeEmail=scannerString.nextLine();

if(!(employeeEmail.matches("^[a-zA-Z0-9+.]+@[a-z+.]+$")))//other pattern for employeeEmail is("[a-zA-Z0-9]+@[a-z]+(.com)")

{

System.out.println("Please enter the valid email ID.Domain name should not contain any numerics or special characters.");

validEmployeeEmail();

}

return employeeEmail;

}

//Overriding the validEmployeeDOB method

public String validEmployeeDOB()

{

System.out.println("Enter the Employee DOB:");

String employeeDOB=scannerString.nextLine();

SimpleDateFormat dob = new SimpleDateFormat("dd/MM/yyyy");

dob.setLenient(false);

try

{

Date dateFormat=dob.parse(employeeDOB);

long dateCalculation=System.currentTimeMillis()-dateFormat.getTime();

long age=(long)((long)dateCalculation/(1000.0\*60\*60\*24\*365));

if(age>=18 && age<=60)

return employeeDOB;

else

{

System.out.println("Please enter the valid date.Example:'DD/MM/YYYY'.Age must be between 18 to 60");

validEmployeeDOB();

}

}

catch(ParseException date)

{

System.out.println("Please enter the valid date."/\*Example:'DD/MM/YYYY'.Age must be between 18 to 60\*/);

validEmployeeDOB();

}

return employeeDOB;

}

public String validEmployeeDOJ()//Overriding the validEmployeeDOJ method

{

System.out.println("Enter the Employee DOJ:");

String employeeDOJ=scannerString.nextLine();

SimpleDateFormat doj = new SimpleDateFormat("dd/mm/yyyy");

doj.setLenient(false);

try

{

Date dateFormat=doj.parse(employeeDOJ);

long dateCalculation=System.currentTimeMillis()-dateFormat.getTime();

if(dateCalculation>0)

return employeeDOJ;

else

{

System.out.println("Please enter the valid date of joining.Example:'DD/MM/YYYY'.No future dates are allowed");

validEmployeeDOJ();

}

}

catch(ParseException date)

{

System.out.println("Please enter the valid date of joining.Example:'dd/MM/yyyy'");

validEmployeeDOJ();

}

return employeeDOJ;

}

//Overriding the validEmployeeMobileNumber method

public String validEmployeeMobileNumber()

{

System.out.println("Enter the employee mobile number:");

String employeeMobileNumber=scannerString.nextLine();

if(!(employeeMobileNumber.matches("(0|91)?[6-9][0-9]{9}")))

{

System.out.println("Please enter the valid mobile number.Mobile number should not contain any alphabets and special characters.");

validEmployeeMobileNumber();

}

return employeeMobileNumber;

}

public void employeeDetails() //CRUD operations

{

// ArrayList<MainEmployee> list=new ArrayList<MainEmployee>();

int selectOption;

do

{

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Employee Management System");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\n1.ADD new employee details\n2.DELETE the employee details\n3.DISPLAY the employee details\n4.UPDATE the employee details\n5.EXIT\n");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Enter the choice to do CRUD operations:");

selectOption=scanner.nextInt();

switch(selectOption)

{

case 1://To ADD the employee Details

{

System.out.println("Enter the number of employee details to be added:");

int numberOfEmployees=scanner.nextInt();

for(int index=1;index<=numberOfEmployees;index++)

{

String employeeID=validEmployeeID();

String employeeName=validEmployeeName();

String employeeEmail=validEmployeeEmail();

String employeeDOB=validEmployeeDOB();

String employeeDOJ=validEmployeeDOJ();

String employeeMobileNumber=validEmployeeMobileNumber();

try

{

String url="jdbc:mysql://localhost:3306/employeemanagement";

String userName="root";

String password="KSP@mysql123";

String query="INSERT INTO employee\_datas(EmployeeID,EmployeeName,EmployeeEmail,EmployeeDOB,EmployeeDOJ,EmployeeMobileNumber) VALUES(?,?,?,?,?,?)";

Class.forName("com.mysql.cj.jdbc.Driver");

Connection connection=DriverManager.getConnection(url,userName,password);

PreparedStatement preparedstatement=connection.prepareStatement(query);

preparedstatement.setString(1, employeeID);

preparedstatement.setString(2, employeeName);

preparedstatement.setString(3, employeeEmail);

preparedstatement.setString(4, employeeDOB);

preparedstatement.setString(5, employeeDOJ);

preparedstatement.setString(6, employeeMobileNumber);

preparedstatement.executeUpdate();

connection.close();

}

catch(Exception e)

{

System.out.println(e);

}

}

System.out.println("\nThe given employee details has been added successfully.");

break;

}

case 2://To DELETE the employee details

{

System.out.println("DELETE the employee details:");

System.out.println("Enter the employee ID to delete:");

String id=scannerString.nextLine();

try

{

String url="jdbc:mysql://localhost:3306/employeemanagement";

String userName="root";

String password="KSP@mysql123";

String query="DELETE FROM employee\_datas WHERE EmployeeID=?";

Class.forName("com.mysql.cj.jdbc.Driver");

Connection connection=DriverManager.getConnection(url,userName,password);

PreparedStatement preparedStatement=connection.prepareStatement(query);

preparedStatement.setString(1,id);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The record with the given employee ID "+id+" is deleted successfully");

}

else

{

System.out.println("The record with the given employee ID "+id+" is not found.");

}

preparedStatement.close();

connection.close();

}

catch(Exception e)

{

System.out.println(e);

}

break;

}

case 3://To DISPLAY the employee details

{

System.out.println("DISPALY the employee details:\n");

System.out.println("Enter the employee ID to display:");

String id=scannerString.nextLine();

try {

String url="jdbc:mysql://localhost:3306/employeemanagement";

String userName="root";

String password="KSP@mysql123";

String query="SELECT \* FROM employee\_datas WHERE EmployeeID=?";

Class.forName("com.mysql.cj.jdbc.Driver");

Connection connection=DriverManager.getConnection(url,userName,password);

PreparedStatement preparedStatement=connection.prepareStatement(query);

preparedStatement.setString(1,id);

ResultSet rs=preparedStatement.executeQuery();

//String name=rs.getString(2);

if(rs.next()==false)

{

System.out.println("\nThe record with the given employee ID "+id+" is not found.\n");

}

else

{

String employeeID=rs.getString(1);

String employeeName=rs.getString(2);

String employeeEmail=rs.getString(3);

String employeeDOB=rs.getString(4);

String employeeDOJ=rs.getString(5);

String employeeMobileNumber=rs.getString(6);

System.out.println(employeeID+":"+employeeName+" "+employeeEmail+" "+employeeDOB+" "+employeeDOJ+" "+employeeMobileNumber);

System.out.println("\nThe record with the given employee ID "+id+" is dispalyed successfully.\n");

}

preparedStatement.close();

connection.close();

}

catch(Exception e)

{

System.out.println(e);

}

break;

}

case 4://To UPDATE the employee details

{

System.out.println("UPDATE the employee details:");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("\n1.UPDATE the Employee ID\n2.UPDATE the Employee Name\n3.UPDATE the Employee Email\n4.UPDATE the Employee DOB\n5.UPDATE the Employee DOJ\n6.UPDATE the Employee Mobile Number\n");

System.out.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.out.println("Enter the choice to update the details:");

int choice=scanner.nextInt();

String employeeID=validEmployeeID();

try

{

String url="jdbc:mysql://localhost:3306/employeemanagement";

String userName="root";

String password="KSP@mysql123";

Class.forName("com.mysql.cj.jdbc.Driver");

Connection connection=DriverManager.getConnection(url,userName,password);

switch(choice)

{

case 1:

{

String newEmployeeID=validEmployeeID();

PreparedStatement preparedStatement=connection.prepareStatement("UPDATE employee\_datas SET EmployeeID=(?) WHERE EmployeeID=(?)");

preparedStatement.setString(1,newEmployeeID);

preparedStatement.setString(2,employeeID);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The old employee ID is updated to new employee ID successfully");

}

else

{

System.out.println("The given employee ID is not found in the database.");

}

break;

}

case 2:

{

String newEmployeeName=validEmployeeName();

PreparedStatement preparedStatement=connection.prepareStatement("UPDATE employee\_datas SET EmployeeName=(?) WHERE EmployeeID=(?)");

preparedStatement.setString(1,newEmployeeName);

preparedStatement.setString(2,employeeID);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The old employee Name is updated to new employee Name successfully");

}

else

{

System.out.println("The given employee ID is not found in the database.");

}

break;

}

case 3:

{

String newEmployeeEmail=validEmployeeEmail();

PreparedStatement preparedStatement=connection.prepareStatement("UPDATE employee\_datas SET EmployeeEmail=(?) WHERE EmployeeID=(?)");

preparedStatement.setString(1,newEmployeeEmail);

preparedStatement.setString(2,employeeID);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The old employee Email is updated to new employee Email successfully");

}

else

{

System.out.println("The given employee ID is not found in the database.");

}

break;

}

case 4:

{

String newEmployeeDOB=validEmployeeDOJ();

PreparedStatement preparedStatement=connection.prepareStatement("UPDATE employee\_datas SET EmployeeDOB=(?) WHERE EmployeeID=(?)");

preparedStatement.setString(1,newEmployeeDOB);

preparedStatement.setString(2,employeeID);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The old employee DOB is updated to new employee DOB successfully");

}

else

{

System.out.println("The given employee ID is not found in the database.");

}

break;

}

case 5:

{

String newEmployeeDOJ=validEmployeeDOJ();

PreparedStatement preparedStatement=connection.prepareStatement("UPDATE employee\_datas SET EmployeeDOJ=(?) WHERE EmployeeID=(?)");

preparedStatement.setString(1,newEmployeeDOJ);

preparedStatement.setString(2,employeeID);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The old employee DOJ is updated to new employee DOJ successfully");

}

else

{

System.out.println("The given employee ID is not found in the database.");

}

break;

}

case 6:

{

String newEmployeeMobileNumber=validEmployeeMobileNumber();

PreparedStatement preparedStatement=connection.prepareStatement("UPDATE employee\_datas SET EmployeeMobileNumber=(?) WHERE EmployeeID=(?)");

preparedStatement.setString(1,newEmployeeMobileNumber);

preparedStatement.setString(2,employeeID);

int i=preparedStatement.executeUpdate();

if(i>0)

{

System.out.println("The old employee MobileNumber is updated to new employee MobileNumber successfully");

}

else

{

System.out.println("The given employee ID is not found in the database.");

}

break;

}

default://When valid choice is not given default block executes

{

System.out.println("Please select a valid option from 1 to 6.The option other than 1 to 6 is not valid");

break;

}

}

}

catch(Exception error)

{

System.out.println(error);

}

break;

}

case 5:

{

System.out.println("EXIT");

break;

}

default://When valid choice is not given default block executes

{

System.out.println("Please select a valid option from 1 to 5.The option other than 1 to 5 is not valid");

break;

}

}

}while(selectOption!=5);

}

}

EmployeeApplication.java

package employee;

public class EmployeeApplication

{

public static void main(String[] args) throws Exception

{

EmployeeDatas employeeDatas = new EmployeeDatas(null,null,null,null,null,null);

employeeDatas.employeeDetails();

}

}