GUJARAT TECHNOLOGICAL UNIVERSITY

L.D. College of Engineering

INTERNSHIP REPORT

Under subject of

SUMMER INTERNSHIP (3170001)

B. E. Semester – VII **I.T. Branch**

Submitted by

Makadiya Grishma Ramnikbhai (190283116006)

Brainy Beam Technologies Pvt. Ltd.

Prof. Jaimin Chavda, Dr.Hiteishi Diwanji
(Faculty Guide) (Head of the I.T.)
Department)

Academic year
(2020-2021)

Company Name: - Brainy Beam Private Limited
Domain: - Core Java
Project Title: - Ahmedabad Live

Tasks:-

- **Data Types:-**

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

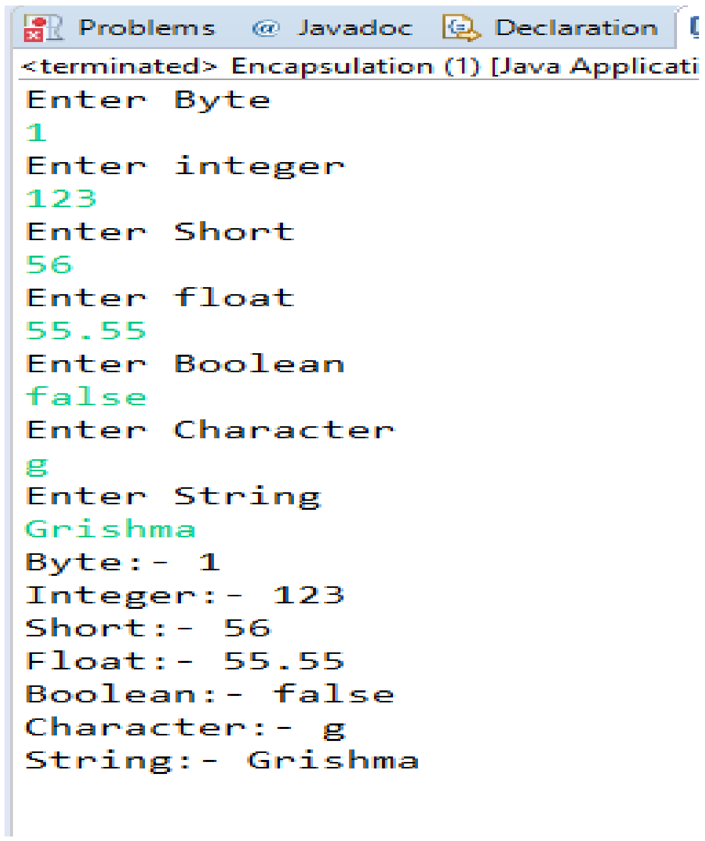
- **Encapsulation:-**
  - **Encapsulation in Java** is a *process of wrapping code and data together into a single unit*, for example, a capsule which is mixed of several medicines.
  - We can create a fully encapsulated class in Java by making all the data members of the class private. Now we can use setter and getter methods to set and get the data in it.
- **Advantage of Encapsulation in Java**
  - It provides you the **control over the data**. Suppose you want to set the value of id which should be greater than 100 only, you can write the logic inside the setter method. You can write the logic not to store the negative numbers in the setter methods.
  - It is a way to achieve **data hiding** in Java because other class will not be able to access the data through the private data members.
  - The encapsulate class is **easy to test**. So, it is better for unit testing.
  - The standard IDE's are providing the facility to generate the getters and setters. So, it is **easy and fast to create an encapsulated class** in Java.

❖ Example:-

```
import java.util.Scanner;
class DataType {

        private byte a;
        private int b;
        private short c;
        private float d;
        private boolean e;
        private char f;
        private String g;
        public byte getA() {
                return a;
        }
        public void setA(byte a) {
                this.a = a;
        }
        public int getB() {
                return b;
        }
        public void setB(int b) {
                this.b = b;
        }
        public short getC() {
                return c;
        }
        public void setC(short c) {
                this.c = c;
        }
        public float getD() {
                return d;
        }
        public void setD(float d) {
                this.d = d;
        }
        public boolean getE() {
                return e;
        }
        public void setE(boolean e) {
                this.e = e;
        }
        public char getF() {
                return f;
        }
        public void setF(char f) {
                this.f = f;
        }
```

```
        public String getG() {
                return g;
        }

        public void setG(String g) {
                this.g = g;
        }
}

public class Encapsulation {
        public static void main(String args[]) {

                DataType DT = new DataType();

                Scanner s = new Scanner(System.in);

                System.out.println("Enter Byte ");
                DT.setA(s.nextByte());

                System.out.println("Enter integer ");
                DT.setB(s.nextInt());

                System.out.println("Enter Short ");
                DT.setC(s.nextShort());

                System.out.println("Enter float");
                DT.setD(s.nextFloat());

                System.out.println("Enter Boolean ");
                DT.setE(s.nextBoolean());

                System.out.println("Enter Character ");
                DT.setF(s.next().charAt(0));

                System.out.println("Enter String ");
                DT.setG(s.next());

                System.out.println("Byte:- " + DT.getA());
                System.out.println("Integer:- " + DT.getB());
                System.out.println("Short:- " + DT.getC());
                System.out.println("Float:- " + DT.getD());
                System.out.println("Boolean:- " + DT.getE());
                System.out.println("Character:- " + DT.getF());
                System.out.println("String:- " + DT.getG());


        }
}
```

OUTPUT:-



```
<terminated> Encapsulation (1) [Java Applicati
Enter Byte
1
Enter integer
123
Enter Short
56
Enter float
55.55
Enter Boolean
false
Enter Character
g
Enter String
Grishma
Byte:- 1
Integer:- 123
Short:- 56
Float:- 55.55
Boolean:- false
Character:- g
String:- Grishma
```

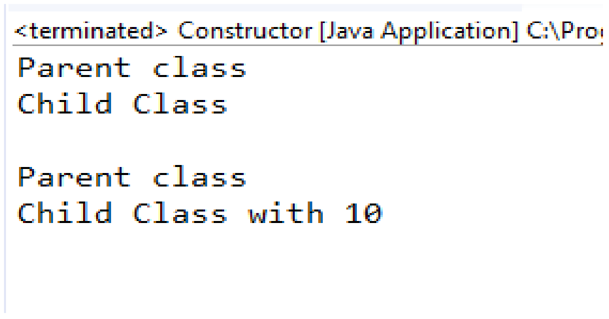
- **Constructor:-**
    - In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.
    - It is a special type of method which is used to initialize the object.
    - Every time an object is created using the new() keyword, at least one constructor is called.
    - It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.
    - There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Example:- In this example we create two constructor default and parameterized and use inheritance. When we call child class constructor then by default parent class default constructor is called.

```
class Parent{
        Parent(){
                System.out.println("Parent class");
        }
        Parent(int a){
                System.out.println("Parent class with "+a);
        }
}
```

```
class Child extends Parent{
        Child(){
                System.out.println("Child Class");
        }
        Child(int a){
                System.out.println("Child Class with "+a);
        }

}
public class Constructor{
        public static void main(String args[]) {
                Child c=new Child();
                System.out.println();
                Child c1=new Child(10);
        }

}
```

OUTPUT:-



```
<terminated> Constructor [Java Application] C:\Pro
Parent class
Child Class

Parent class
Child Class with 10
```

NOTE: - When we call parameterized constructor of child class then it's call default constructor instead of parameterized constructor of parent class.

<u>ACCESS MODIFIRES:-</u>

- The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class. We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.
- There are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.

3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

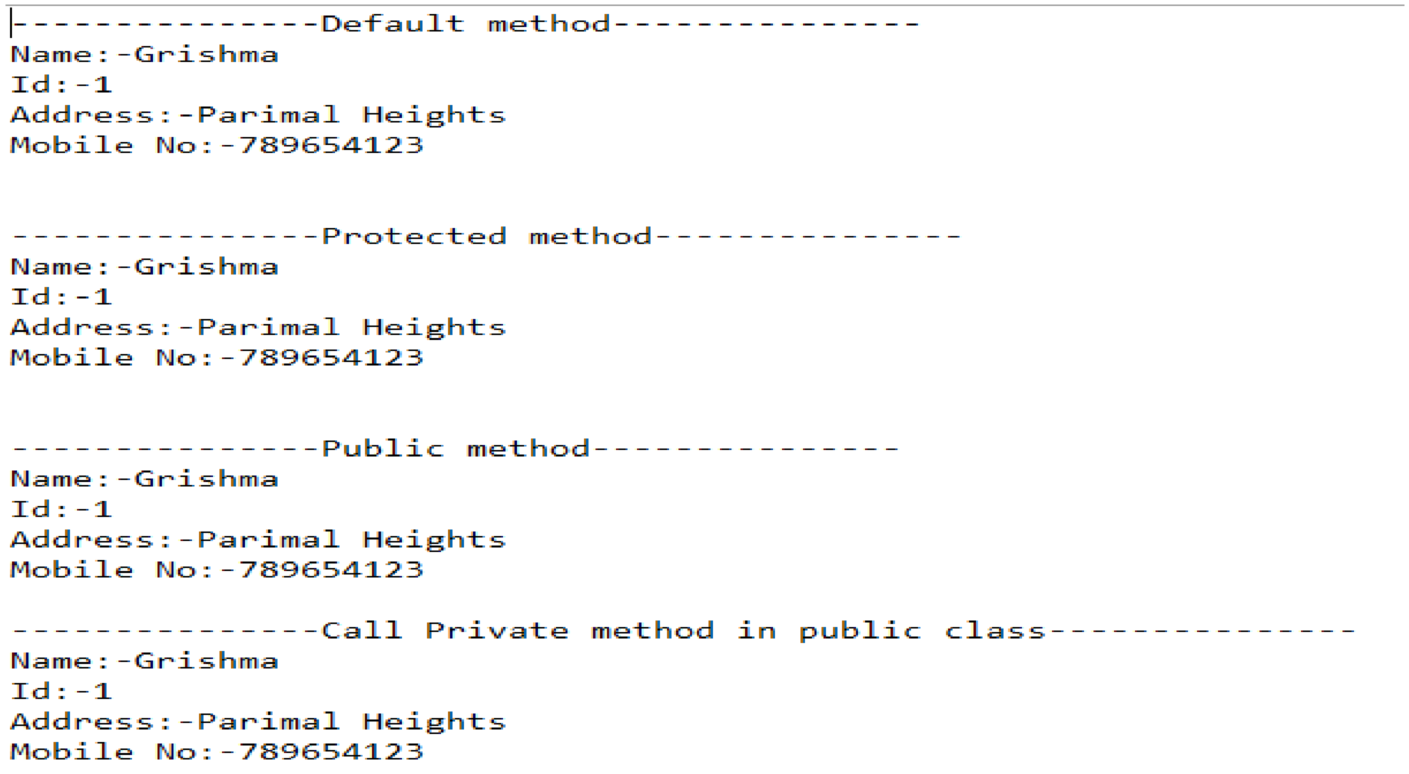
Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

Example:-

```
class Access{
	private int id=1;
	public String name="Grishma";
	protected String addrees="Parimal Heights";
	long mobileNo = 789654123;

	private void accessPrivate(){
		System.out.println("Name:-"+name);
		System.out.println("Id:-"+id);
		System.out.println("Address:-"+addrees);
		System.out.println("Mobile No:-"+mobileNo);
	}
	public void accessPublic(){
		System.out.println("Name:-"+name);
		System.out.println("Id:-"+id);
		System.out.println("Address:-"+addrees);
		System.out.println("Mobile No:-"+mobileNo);
		System.out.println("\n---------------Call Private method in public class-----------
----");
		accessPrivate();
	}
	protected void accessProtected(){
		System.out.println("Name:-"+name);
		System.out.println("Id:-"+id);
		System.out.println("Address:-"+addrees);
```

```
                System.out.println("Mobile No:-"+mobileNo);
            }
            void accessDefault(){
                System.out.println("Name:-"+name);
                System.out.println("Id:-"+id);
                System.out.println("Address:-"+addrees);
                System.out.println("Mobile No:-"+mobileNo);
            }
        }
        public class AccessModifier {

            public static void main(String args[]) {
                Access a=new Access();
                System.out.println("---------------Default method---------------");
                a.accessDefault();
                System.out.println("\n\n---------------Protected method---------------");
                a.accessProtected();
                System.out.println("\n\n---------------Public method---------------");
                a.accessPublic();
            }
        }
```

OUTPUT:-



```
---------------Default method---------------
Name:-Grishma
Id:-1
Address:-Parimal Heights
Mobile No:-789654123


---------------Protected method---------------
Name:-Grishma
Id:-1
Address:-Parimal Heights
Mobile No:-789654123


---------------Public method---------------
Name:-Grishma
Id:-1
Address:-Parimal Heights
Mobile No:-789654123

---------------Call Private method in public class---------------
Name:-Grishma
Id:-1
Address:-Parimal Heights
Mobile No:-789654123
```

In this example we use all access modifier we don't use private method or member outside the class so we use that in public class so using public method we can use private method or member outside the class because of that we can't directly change the value on private member or method.

INHERITANCE:-

- **Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object. It is an important part of OOPs (Object Oriented programming system).
- The idea behind inheritance in Java is that you can create new classes that are built upon existing classes. When you inherit from an existing class, you can reuse methods and fields of the parent class. Moreover, you can add new methods and fields in your current class also.
- Inheritance represents the **IS-A relationship** which is also known as a *parent-child* relationship.
- We use inheritance For **Method Overriding** (so runtime polymorphism can be achieved) and for **Code Reusability**.

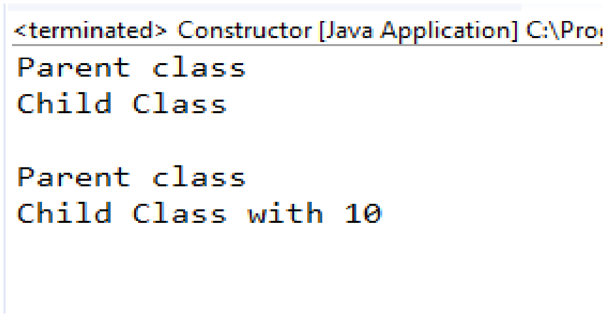
- **Types Of Inheritance:-**

  1. Single Inheritance:-

  - When a class inherits another class, it is known as a single inheritance.

```
class Parent{
        Parent(){
                System.out.println("Parent class");
        }
        Parent(int a){
                System.out.println("Parent class with "+a);
        }
}
class Child extends Parent{
        Child(){
                System.out.println("Child Class");
        }
        Child(int a){
                System.out.println("Child Class with "+a);
        }

}
public class Constructor{
        public static void main(String args[]) {
                Child c=new Child();
                System.out.println();
                Child c1=new Child(10);
        }

}
```

OUTPUT:-



```
<terminated> Constructor [Java Application] C:\Pro
Parent class
Child Class

Parent class
Child Class with 10
```

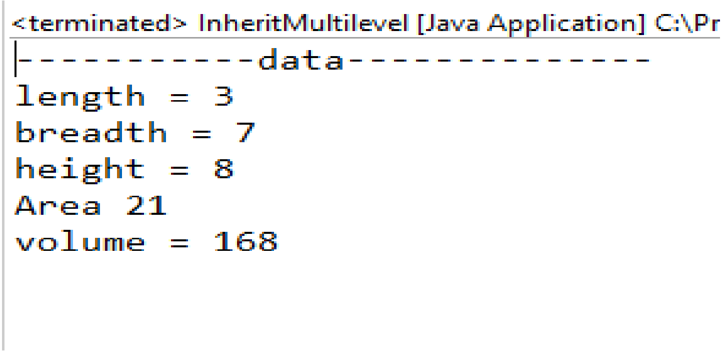
2. Multi-Level Inheritance:-

- When there is a chain of inheritance, it is known as multilevel inheritance.

```
class Data1 {
	int l, b;

	Data1(int l, int b) {
		System.out.println("-----------data--------------");
		this.l = l;
		this.b = b;
		System.out.println("length = " + this.l + "\nbreadth = " + this.b);
	}
}

class Area1 extends Data1 {
	int area;

	Area1(int l, int b) {
		super(l, b);
	}

	void fun_area() {
		area = super.l * super.b;
		System.out.println("Area " + area);
	}
}

class Volume1 extends Area1 {
	int h, vol;

	Volume1(int l, int b, int h) {
		super(l, b);
		this.h = h;
		System.out.println("height = " + h);
	}
```

```
		void fun_vol() {
			vol = super.l * super.b * h;
			System.out.println("volume = " + vol);
		}
	}

	class InheritMultilevel {
		public static void main(String args[]) {
			Volume1 v = new Volume1(3, 7, 8);
			v.fun_area();
			v.fun_vol();
		}
	}
```

OUTPUT:-



```
<terminated> InheritMultilevel [Java Application] C:\Pr
-----------data--------------
length = 3
breadth = 7
height = 8
Area 21
volume = 168
```

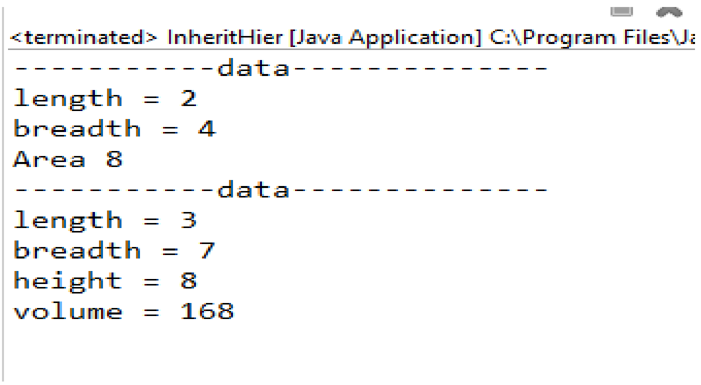
3. Hierarchical Inheritance

- When two or more classes inherits a single class, it is known as *hierarchical inheritance*.

```
class Data {
	int l, b;

	Data(int l, int b) {
		System.out.println("-----------data--------------");
		this.l = l;
		this.b = b;
		System.out.println("length = " + this.l + "\nbreadth = " + this.b);
	}
}
class Area extends Data {
	int area;

	Area(int l, int b) {
		super(l, b);
	}
```

```
        void fun_area() {
                area = super.l * super.b;
                System.out.println("Area " + area);
        }
}
class Volume extends Data {
        int h, vol;

        Volume(int l, int b, int h) {
                super(l, b);
                this.h = h;
                System.out.println("height = " + h);
        }
        void fun_vol() {
                vol = super.l * super.b * h;
                System.out.println("volume = " + vol);
        }
}
class InheritHier {
        public static void main(String args[]) {
                Area a = new Area(2, 4);
                a.fun_area();
                Volume v = new Volume(3, 7, 8);
                v.fun_vol();
        }
}
```

OUTPUT:-



```
<terminated> InheritHier [Java Application] C:\Program Files\J
-----------data--------------
length = 2
breadth = 4
Area 8
-----------data--------------
length = 3
breadth = 7
height = 8
volume = 168
```

NOTE: - Multiple and Hybrid inheritance are not implemented by default in java. They are implemented through interface.

INTERFACE:-

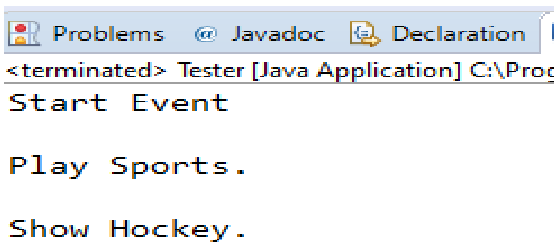
- An **interface in Java** is a blueprint of a class. It has static constants and abstract methods.
- The interface in Java is *a mechanism to achieve* abstraction. There can be only abstract methods in the Java interface, not method body. It is used to achieve abstraction and multiple inheritance in Java.
- In other words, you can say that interfaces can have abstract methods and variables. It cannot have a method body.
- Java Interface also **represents the IS-A relationship.**

Multiple Inheritance:-

- Multiple inheritance is not supported in java but using interfaces we can implement multiple inheritance.

```
interface Event {
  public void start();
}
interface Sports {
  public void play();
}
interface Hockey extends Sports, Event{
  public void show();
}
public class Tester{
  public static void main(String[] args){
    Hockey hockey = new Hockey() {
      public void start() {
        System.out.println("Start Event\n");
      }
      public void play() {
        System.out.println("Play Sports.\n");
      }
      public void show() {
        System.out.println("Show Hockey.\n");
      }
    };

    hockey.start();
    hockey.play();
    hockey.show();
  }
}
```

OUTPUT:-



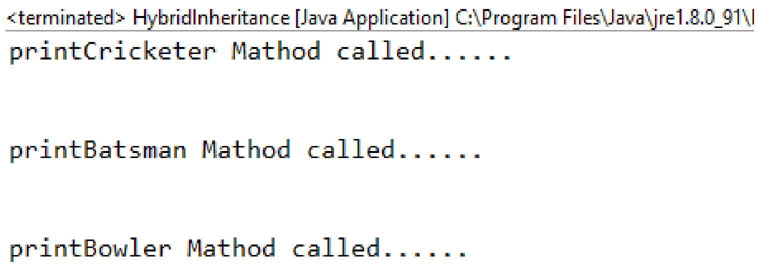
```
<terminated> Tester [Java Application] C:\Proc
Start Event

Play Sports.

Show Hockey.
```

Hybrid Inheritance:-

- Hybrid inheritance is a combination of more than one type of inheritance. Here we combine hierarchical and multiple inheritance.

```
interface Cricketer {
        public void printCricketer();
}

interface Bowler extends Cricketer {
        public void printBowler();
}

interface Batsman extends Cricketer {
        public void printBatsman();
}

class AllRounder implements Bowler, Batsman {

        @Override
        public void printCricketer() {
                System.out.println("printCricketer Mathod called......\n\n");

        }

        @Override
        public void printBatsman() {
                System.out.println("printBatsman Mathod called......\n\n");

        }

        @Override
        public void printBowler() {
                System.out.println("printBowler Mathod called......\n\n");

        }
}

public class HybridInheritance {
        public static void main(String args[]) {
                AllRounder ar = new AllRounder();
                ar.printCricketer();
                ar.printBatsman();
                ar.printBowler();
        }
}
```

OUTPUT:-



```
<terminated> HybridInheritance [Java Application] C:\Program Files\Java\jre1.8.0_91\I
printCricketer Mathod called......


printBatsman Mathod called......


printBowler Mathod called......
```

PROJECT: - Ahmedabad Live

Here we simply register and login our-self through console. Here we can register only single user. In this program we use multiple concept like access modifiers, inheritance, looping statements etc…

ApplicationContext.java :-

```
	package com.brainy_beam.ahmlive.context;

	public class ApplicationContext {

		private static String firstName;
		private static String lastName;
		private static String email;
		private static String password;
		private static String confirmPassword;
		private static String address;
		private static long mobileNo;

		public static String getFirstName() {
			return firstName;
		}

		public static void setFirstName(String firstName) {
			ApplicationContext.firstName = firstName;
		}

		public static String getLastName() {
			return lastName;
		}

		public static void setLastName(String lastName) {
			ApplicationContext.lastName = lastName;
		}

		public static String getEmail() {
			return email;
		}

		public static void setEmail(String email) {
			ApplicationContext.email = email;
		}

		public static String getPassword() {
			return password;
		}

		public static void setPassword(String password) {
			ApplicationContext.password = password;
		}

		public static String getConfirmPassword() {
			return confirmPassword;
		}

		public static void setConfirmPassword(String confirmPassword) {
			ApplicationContext.confirmPassword = confirmPassword;
		}

		public static String getAddress() {
			return address;
		}

		public static void setAddress(String address) {
			ApplicationContext.address = address;
		}

		public static long getMobileNo() {
			return mobileNo;
		}

		public static void setMobileNo(long mobileNo) {
			ApplicationContext.mobileNo = mobileNo;
		}
	}
```

ApplicationStartup.java:-

```
	package com.brainy_beam.ahmlive.main;

	import java.util.Scanner;

	public class ApplicationStartup {

		public static void main(String[] args) {
			System.out.println("................Welcome to Ahmedabad Live.............\n\n\n");

			Scanner s = new Scanner(System.in);
			int choice;

			do {
				System.out.println(" Enter 1 for Registration \n Enter 2 for Login \n
	Enter 3 for exit\n");
				System.out.println("Enter your choice:-");
				choice = s.nextInt();

				switch (choice) {
				case 1:
					UserRegistration userRegister = new UserRegistration();
					if (userRegister.register()) {
						System.out.println("Register Successfully........\n\n");

					} else {
						System.out.println("Registration Unsuccessful\n");
					}

				case 2:
					UserLogin userLogin = new UserLogin();
					while (!userLogin.login()) {
						System.out.println("Invalid email or password.........\n\n");
					}
					System.out.println("............Login Successful........\n\n");

				case 3:
					System.out.println("----------Thank you--------");
					break;

				default:
					System.out.println("Please enter valid choice.....");
				}
			} while (choice != 3);
		}
	}
```

UserRegistration.java:-

```
	package com.brainy_beam.ahmlive.main;

	import java.util.Scanner;
	import java.util.function.ToLongFunction;

	import com.brainy_beam.ahmlive.context.ApplicationContext;
	import com.brainy_beam.ahmlive.util.ValidateInput;
```

```java
public class UserRegistration {

	public boolean register() {
		Scanner s = new Scanner(System.in);

		System.out.println("Enter First Name:-");
		String firstName = s.next();
		if (ValidateInput.validateString(firstName)) {
			ApplicationContext.setFirstName(firstName);
		} else {
			return false;
		}

		System.out.println("Enter Last Name:-");
		String lastName = s.next();
		if (ValidateInput.validateString(lastName)) {
			ApplicationContext.setLastName(lastName);
		} else {
			return false;
		}

		System.out.println("Enter Mobile Number:-");
		long mobileNo = s.nextLong();
		if (ValidateInput.validateNumber(mobileNo)) {
			ApplicationContext.setMobileNo(mobileNo);
		} else {
			return false;
		}

		System.out.println("Enter Email:-");
		String email = s.next();
		if (ValidateInput.validateString(email)) {
			ApplicationContext.setEmail(email);
		} else {
			return false;
		}

		System.out.println("Enter Password:-");
		ApplicationContext.setPassword(s.next());

		System.out.println("Enter Confirm Password:-");
		ApplicationContext.setConfirmPassword(s.next());
```

```
                System.out.println("Enter Address:-");
                String address=s.next();
                if (ValidateInput.validateString(address)) {
                        ApplicationContext.setAddress(address);
                } else {
                        return false;
                }

                System.out.println("You are Registered...... \n\n");

                return true;
        }
}
```

UserLogin.java:-

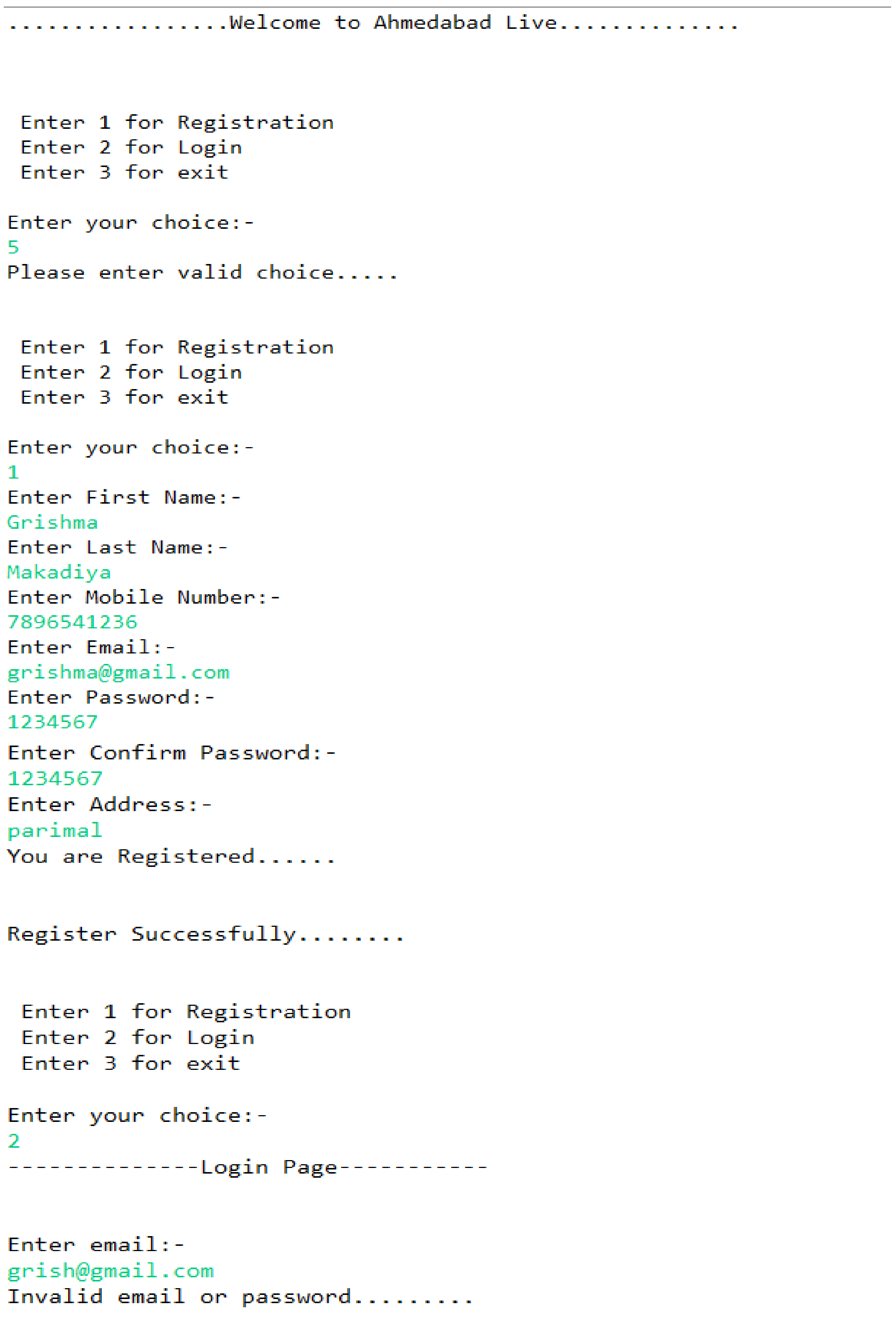
```
package com.brainy_beam.ahmlive.main;

import java.applet.AppletContext;
import java.util.Scanner;

import com.brainy_beam.ahmlive.context.ApplicationContext;
import com.brainy_beam.ahmlive.util.ValidateInput;

public class UserLogin {

        public boolean login() {
                Scanner s=new Scanner(System.in);
                boolean isValidate;

                System.out.println("--------------Login Page-----------\n\n");

                System.out.println("Enter email:-");
                String email = s.next();

                isValidate=ValidateInput.validateString(email);
                if (isValidate && ApplicationContext.getEmail().equals(email)) {
                        isValidate=true;
                } else {
                        return false;
                }

                System.out.println("Enter password");
                String password=s.next();
                isValidate=ValidateInput.validateString(password);
```

```
				if (isValidate && ApplicationContext.getPassword().equals(password)) {
					isValidate=true;
				} else {
					return false;
				}
				return isValidate;
			}

		}
```

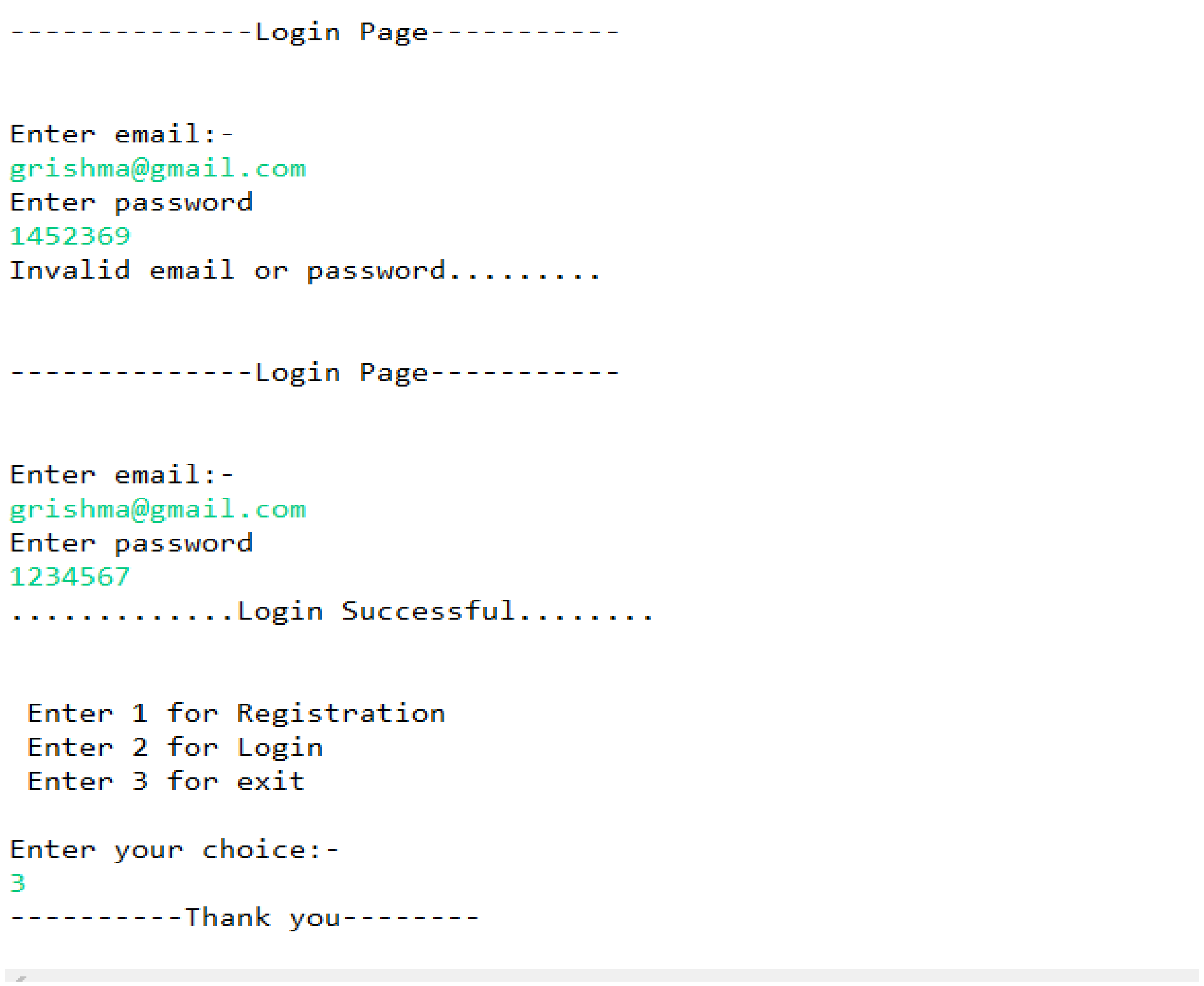
ValidateInput.java:-

```
		package com.brainy_beam.ahmlive.util;

		public class ValidateInput {
			public static boolean validateString(String value) {
				if (value != null && !("".equals(value))) {
					return true;
				} else {
					return false;
				}
			}

			public static boolean validateNumber(long number) {
				if (number != 0 && number > 600000000) {
					return true;
				} else {
					return false;
				}
			}

		}
```

OUTPUT:-



```
.................Welcome to Ahmedabad Live..............



 Enter 1 for Registration
 Enter 2 for Login
 Enter 3 for exit

Enter your choice:-
5
Please enter valid choice.....


 Enter 1 for Registration
 Enter 2 for Login
 Enter 3 for exit

Enter your choice:-
1
Enter First Name:-
Grishma
Enter Last Name:-
Makadiya
Enter Mobile Number:-
7896541236
Enter Email:-
grishma@gmail.com
Enter Password:-
1234567
Enter Confirm Password:-
1234567
Enter Address:-
parimal
You are Registered......


Register Successfully........


 Enter 1 for Registration
 Enter 2 for Login
 Enter 3 for exit

Enter your choice:-
2
--------------Login Page-----------


Enter email:-
grish@gmail.com
Invalid email or password.........
```



```
-------------Login Page-----------


Enter email:-
grishma@gmail.com
Enter password
1452369
Invalid email or password.........


-------------Login Page-----------


Enter email:-
grishma@gmail.com
Enter password
1234567
.............Login Successful........


 Enter 1 for Registration
 Enter 2 for Login
 Enter 3 for exit

Enter your choice:-
3
----------Thank you--------
```

EXCEPTIONS:-

- **Dictionary Meaning:** Exception is an abnormal condition.
- In Java, an exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

EXCEPTION HANDLING:-

- Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.
- The core advantage of exception handling is to maintain the normal flow of the application. An exception normally disrupts the normal flow of the application that is why we use exception handling.

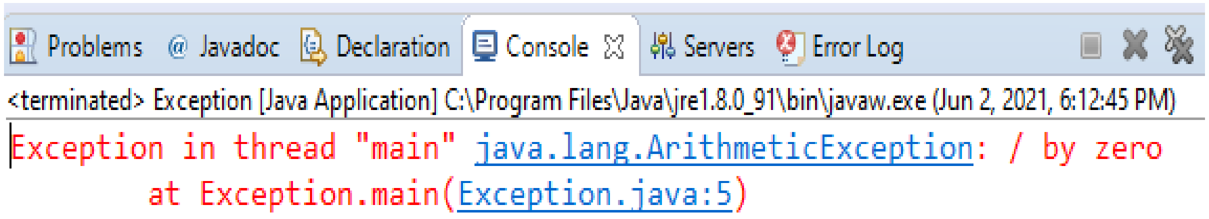
➢ There are 5 keywords which are used in handling exceptions in Java.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place exception code. The try block must be followed by either catch or finally. It means, we can't use try block alone.
catch	The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later.
finally	The "finally" block is used to execute the important code of the program. It is executed whether an exception is handled or not.
throw	The "throw" keyword is used to throw an exception.
throws	The "throws" keyword is used to declare exceptions. It doesn't throw an exception. It specifies that there may occur an exception in the method. It is always used with method signature.

With Out Exception handling

```
public class Exception {
	public static void main(String args[]) {

		int data = 100 / 0;

		System.out.println(data);
	}
}
```

OUTPUT:-

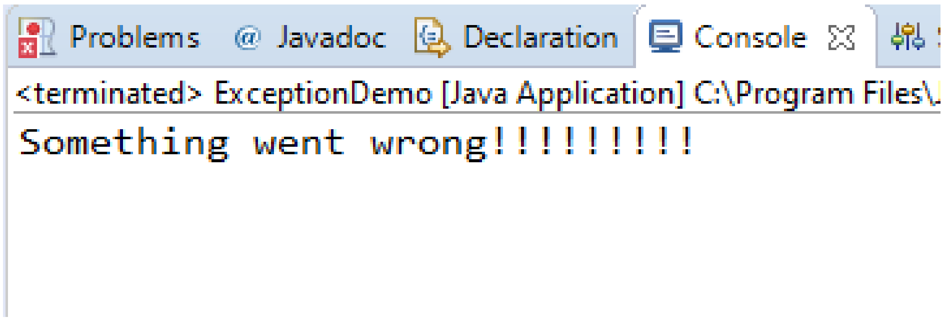


```
<terminated> Exception [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe (Jun 2, 2021, 6:12:45 PM)
Exception in thread "main" java.lang.ArithmeticException: / by zero
	at Exception.main(Exception.java:5)
```

This code throw exception because we divide the 100 by 0. Here we know that this code throw exception but in real time scenario we don’t know what values are giving by user so avoid this exception we use exception handling.
Now we use try catch for handling that exception.

```
public class ExceptionDemo {
	static int data;
	public static void main(String args[]) {
		try {
			data = 100 / 0;
			System.out.println(data);
		} catch (Exception e) {
			System.out.println("Something went wrong!!!!!!!!!");
		}
	}
}
```

OUTPUT:-

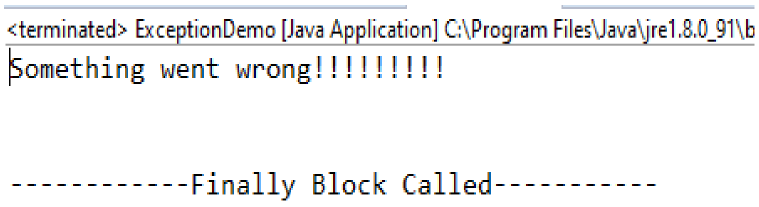


```
<terminated> ExceptionDemo [Java Application] C:\Program Files\
Something went wrong!!!!!!!!!
```

In above example, we show that if any exception is occurred then after the code of the line when exception is occurred are not execute. If in some situation we want to show important message to user or perform some important code like connection close, so in that situation we use final block with try or try catch block. Final block is always performed in any situation. In next example we show try finally block.

```
	public class ExceptionDemo {
		static int data;
		public static void main(String args[]) {
			try {
				data = 100 / 0;
				System.out.println(data);
			} catch (Exception e) {
				System.out.println("Something went wrong!!!!!!!!!\n\n");
			}

			finally {
				System.out.println("------------Finally Block Called-----------");
			}
		}
	}
```

OUTPUT:-



```
<terminated> ExceptionDemo [Java Application] C:\Program Files\Java\jre1.8.0_91\b
Something went wrong!!!!!!!!!


------------Finally Block Called-----------
```