Practice project -2

1.Java Program for Implicit and Explicit typecasting:

a. Implicit Typecasting

```
package com.simplilearn.typecasting; public class ImplicitTypeCasting { public static void main(String[] args) {

        byte smallCount = 100 ;
    short shortCount = smallCount ;
    int intCount = shortCount ;
    long bigCount = intCount
        float floatCount = bigCount;
double decimalCount = bigCount;
double decimalCount2 = floatCount;
        System.out.println("Byte count : "+ smallCount);
        System.out.println("Integer count : "+ intCount);
        System.out.println("Short count : "+ shortCount);
        System.out.println("Long count : "+ bigCount);
        System.out.println("Float count : "+ floatCount);
        System.out.println("Doube count 1 : "+ decimalCount);
        System.out.println("Double count 2 : "+ decimalCount2);
    }

}
```

b.Explicit typecasting:

```
package com.simplilearn.typecasting; public class ExplicitTypeCast {
```

```
        public static void main(String[] args) {               double accBalance =
100.4545d;   // 8 byte => decimal point value                  long bigBalance = (long)
accBalance;                int intBalance = (int) accBalance;                    short
shortBalance = (short) bigBalance;                  byte byteBalance = (byte)
accBalance;

                    System.out.println("Acount Balance (double) : "+ accBalance);
                     System.out.println("Acount Balance (long) : "+ bigBalance);
                     System.out.println("Acount Balance (int) : "+ intBalance);
                     System.out.println("Acount Balance (short) : "+ shortBalance);
                     System.out.println("Acount Balance (byte) : "+ byteBalance);

        }
}
```

2.Access Modifiers:

a.Public Access Modifier:

```
package com.simplilearn.accessmodifier;
public class PublicAccessModifier {    public
static void main(String[] args) {
                    Park park = new Park();
                    System.out.println(park.title);
                    System.out.println(park.statuesCount);                System.out.println("-
----------");
                    park.showDetails();
park.showStatues();

  }
}
```

```
class Park {
        public String title = "Public Health Center Park";
        public int chairCount = 100;
public short statuesCount = 30;
public byte games = 10;         public
void showDetails() {
 System.out.println(this.title + " has chair count : "+ this.chairCount +" and statues are :
"+this.statuesCount);
        }
        public void showStatues() {
                System.out.println(this.title + " has total "+this.statuesCount +" statues.");
        }

        public Park() { };
}
```

b. Private Access Modifier:

```
package com.simplilearn.accessmodifier;
public class PrivateAccessModifier {
public static void main(String[] args) {

                BankAccount account = new BankAccount();
        account.showName();
account.showBalance();
        }
}

class BankAccount {
```

```
	private long accNo = 349583348;
private double accBalance = 45454.454d;
private String fullName = "Harika Dodda";
private String email = "harika@gmail.com";
private String showEmail() {
		return this.email;
	}

	public void showBalance() {
		System.out.println("The Account : " + this.accNo + " has balance :
$"+this.accBalance);
	}

	public void showName() {
		System.out.println("The Account : " + this.accNo + " blongs to : "+this.fullName);
	}

	public BankAccount() { }

	// private BankAccount() { }  // Private Constructor based class can not be instantiated.


}
```

c.Protected Access Modifier:

```
package com.simplilearn.accessmodifier; import
com.simplilearn.typecasting.ExtendedHouse; public class
ProtectedAccessModifer extends ExtendedHouse{
```

```
	public static void main(String[] args) {
		House house = new House();
		System.out.println("Name : " + house.name);
		System.out.println("No. of Room's : " + house.noOfRooms);
		System.out.println("No. of Kitechen's : " + house.noOfKitechen);
		System.out.println("-----------------");
		house.showDetails();
		ExtendedHouse extendedHouse = new ExtendedHouse();
		System.out.println("-----------------");

		ProtectedAccessModifer accessModifer = new ProtectedAccessModifer();
		System.out.println(accessModifer.area);
System.out.println(accessModifer.price);		accessModifer.showAreaAndPrice();
	}
}

class House {

	protected String name = "White House";
protected byte noOfRooms = 20;		protected
byte noOfKitechen = 5;  protected byte
noOfBalcony = 15;

	protected String showName() {
		return this.name;
	}

	protected void showDetails() {
		System.out.println(this.name + " has total "+this.noOfRooms + " rooms ,
"+this.noOfKitechen
				+ " kitchen and "+this.noOfBalcony + " balconines");
```

```
	}


	protected House() {}
```

d.Default Access Modifier:

```
package com.simplilearn.accessmodifier; public
class DefaultAccessmodifier {


	public static void main(String[] args) {
		FarmHouse farmHouse = new FarmHouse();


		System.out.println("Title : " + farmHouse.title);
		System.out.println("Price : " + farmHouse.price);


		farmHouse.showDetails();
farmHouse.showPrice();
	}
}



class FarmHouse {


	String title = "Open Green Farm";
int chairCount = 100;	byte games = 10;
	float price = 4854.45f;


	void showDetails() {
 System.out.println(this.title + " has chair count : "+this.chairCount + " and per day price is :
$"+this.price );
	}
```

```
	void showPrice() {
		System.out.println(this.title + " per day price is : $"+this.price );
	}

	FarmHouse() {} // default constructor : A constructor without any arguments or with the default value for every argument

}
```

3. While Loop:

```
package com.simplilearn.loop; public class
WhileLoop {	public static void
main(String[] args) {
 int count = 1;	while(count<=10) {
			System.out.println("Count : "+ count);
			// breaking count
			count++;
		}

		System.out.println("----------------------");
		// reverse counter
	int revCount = 10;
while(revCount>=1) {
			System.out.println("Count : "+ revCount);
			// breaking count
			revCount--;
		}

		System.out.println("----------------------");
```

```
		// infinity loop
		while(true) {
			System.out.println("infinitive while loop");
		}
	}

}
```

4.Do While Loop:

```
package com.simplilearn.loop; public
class DoWhileLoop {

	public static void main(String[] args) {
		int count = 1;
		do {

			System.out.println("Count : " + count);
			// breaking condition
			count++;
		} while (count <= 10);



		System.out.println("----------------------");
// create a reverse counter
		int revCount = 10;
		do {

			System.out.println("Count : " + revCount);
			// breaking condition
			revCount--;
```

```
                } while (revCount >= 1);



                System.out.println("----------------------");
                //infinity loop
                do {

                        System.out.println("Infinative Do while");
                } while (true);
        }

}
```

5. For Loop:

```
package com.simplilearn.loop; public class
ForLoop {         public static void
main(String[] args) {
                for (int count = 1; count <= 10; count++) {
                        System.out.println("Count : " + count);
                }
                System.out.println("----------------------");
 // create a reverse counter // for(initialization; condition; increment/decrement)   for (int revCount
= 10; revCount >= 1 ; revCount--) {
                        System.out.println("Count : " + revCount);
                }
                System.out.println("----------------------");
                // infinity for loop
                for (int revCount = 10; true ; revCount--) {
                        System.out.println("Count : " + revCount);
                }
```

```
    }

}
```

6. Demonstrating Class,Objects,Constructors:

Class:

In Java, a class is a blueprint or a template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have. Classes are used to model real-world entities by encapsulating their characteristics and functionalities.

Objects:

An object is an instance of a class. It is a tangible entity that is created based on the blueprint defined by a class. Objects have states (values of properties) and behaviors (methods) associated with them.

Constructors:

A constructor is a special method within a class that is responsible for initializing the properties of an object when it is created. Constructors have the same name as the class and do not have a return type.

Example: package

com.simplilearn.classdemo; public

class TrainingApp {

```
    public static void main(String[] args) {
        Trainer trainer1 = new Trainer("John Doe", 30, "john@gmail.com", "Automation
Test");
  Trainer trainer2 = new Trainer("Venkata", 30, "venkata@gmail.com", "Java Programming");
        Trainer trainer3 = new Trainer("Sasidhar", 30, "sasidhar@gmail.com", "QA Test");
  Trainer trainer4 = new Trainer("Akshata", 30, "akshata@gmail.com", "Java Programming");

        trainer1.introduce();
trainer1.conductTraining();
```

```
                trainer2.introduce();
trainer3.introduce(); trainer4.introduce();
        }
}

class Trainer  {

 //data  members   String
name;  int age;
        String email;
        String expertise;

        // get method
        public String getName() {
                return name;
        }

        // other logical method
public void introduce() {
 System.out.println("Hi, I'm " + this.name + ", a trainer with expertise in " + this.expertise + ".");
        }

        public void conductTraining() {
                System.out.println(this.name + " is conducting a training session.");
        }

        Trainer(String name, int age, String email, String expertise) {
                this.name = name;
this.age = age;             this.email =
```

email; this.expertise =

expertise;

    }

}

7.Demonstrating Inheritance:

Inheritance is a fundamental concept in object-oriented programming that allows a new class to inherit the properties and behaviors of an existing class .

There are different types of Inheritance. Thay are as follows:

1.Single Inheritance:

- In Single Inheritance, a subclass inherits from only one superclass.
- It forms, a linear or hierarchical relationship between classes.

2.Multiple Inheritance:

- Multiple inheritance allows a subclass to inherit from more than one superclass.
- While it can provide increased functionality, it can lead to the diamond problem.
- Java doesn't support multiple inheritance directly for classes to avoid the diamond problem. However, it supports multiple inheritance through interfaces.

3. Multi-Level Inheritance:

- Multilevel inheritance involves a chain of inheritance with more than two levels.
- A class serves as a superclass for another class, which, in turn, becomes the superclass for another class.

4. Hierarchical Inheritance:

- In hierarchical inheritance, multiple subclasses inherit from a single superclass.
- It forms like a tree structure.

5. Hybrid Inheritance:

- Hybrid inheritance is a combination of two or more types of inheritance within a single program.
- It can be a combination of any of the above types.

8. collections:

package com.simplilearn.collection.list.employee;

```
import java.util.Iterator; import
java.util.LinkedList; import
java.util.List;

public class EmployeeList {


	public static void main(String[] args) {
		List<Employee> employees = new LinkedList<Employee>();
employees.add(new Employee(1000, "John Doe", 50000));		employees.add(new
Employee(1001, "Mike Smith", 400233));		employees.add(new Employee(1002,
"Sangeetha V", 78000));		employees.add(new Employee(1003, "Tharun
Venkata", 56566));
		System.out.println(employees);
		System.out.println(employees.get(2));
		System.out.println("------------------");
Iterator<Employee> itr = employees.iterator();		while
(itr.hasNext()) {
			System.out.println(itr.next());
		}


		System.out.println("-----------------");
		// enhance for loop
for (Employee emp : employees) {
			System.out.println(emp);
		}
	}
}


class Employee {
```

```
	public int empId;
public String empName;
public double salary;

	public Employee() {
	};

	public Employee(int empId, String empName, double salary) {
		super();
		this.empId = empId;
this.empName = empName;		this.salary
= salary;
	}

	@Override
	public String toString() {
  return "Employee (empId=" + this.empId + ", empName=" + this.empName + ", salary= $" +
this.salary + ")";
	};
}
```

9. Try-catch block:

```
package com.simplilearn.exceptionhandling; public
class ExceptionHandlingDemo {

	public static void main(String[] args) {

		System.out.println("--- Program started ! ---");
		try {
```

```
                int accountBalance = 5000;
            int intrestPer = 0 ;

                int totalIntrest = ( accountBalance / intrestPer ) * 100;

                System.out.println("Total Amount "+ totalIntrest);

            } catch (ArithmeticException e) {
                System.out.println("Exception Occured and handled ::  "+e.getClass());
                System.out.println("Exception Message :: "+e.getMessage());
            }
            System.out.println("--- Program ended ! ---");
        }
}
```

10.Throw and throws Keyword:

```
package com.simplilearn.exceptionhandling;
import java.util.Scanner; class
InvalidSalaryException extends Exception {
public InvalidSalaryException(String message) {
super(message);
   }
}
class Employee {
private String name;
private double salary;

   public Employee(String name, double salary) throws InvalidSalaryException {
this.name = name;
      if (salary < 0) {
```

```
            throw new InvalidSalaryException("Invalid salary: Salary cannot be negative.");
        }
        this.salary = salary;
    }

    public void displayDetails() {
        System.out.println("Employee Details:");
        System.out.println("Name: " + name);
        System.out.println("Salary: $" + salary);
    }
}

public class EmployeeSalaryDetails {
public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
         try
{
            System.out.print("Enter employee name: ");
            String name = scanner.nextLine();

            System.out.print("Enter employee salary: $");
double salary = Double.parseDouble(scanner.nextLine());
            Employee employee = new Employee(name, salary);

            employee.displayDetails();
        } catch (NumberFormatException e) {
            System.out.println("Invalid input for salary. Please enter a numeric value.");
        } catch (InvalidSalaryException e) {
            System.out.println("Error: " + e.getMessage());
```

```
    } finally {
scanner.close();
    }
  }
}
```

11. Try block with parameters:

```
package com.simplilearn.exceptionhandling; import
java.util.Scanner;

public class ExceptionHandling { public static
        void main(String[] args) {
                calcintrest();
        }
        private static void calcIntrest(int intAmt ) {
                try {
                        int balance = 3000;
                        int total  = (balance / intAmt ) * 100;
                        System.out.println("Total amount :: "+ total);

                } catch (ArithmeticException e) {
                        System.out.println("Exception Occures : " + e.getClass());
                        System.out.println("Exception Message : " + e.getMessage());
                }
        }
```

12.Multiple Catch Blocks:

```
package com.simplilearn.exceptionhandling;
public class ExceptionHandlingDemo2
public static void main(String[] args) {
                calcIntrest(30);
calcIntrest(0);             calcIntrest(40);

                System.out.println("--------------");
strLengthCalculator("Today is a good day !");
strLengthCalculator(""); strLengthCalculator(null); strLengthCalculator("Hello
EveryOne");

                System.out.println("--------------");
multiConvertor("5000", 500);
                multiConvertor("Today is a good day", 500);
        }

        private static void calcIntrest(int intAmt ) {
                try {
                        int balance = 3000;
                        int total  = (balance / intAmt ) * 100;
                        System.out.println("Total amount :: "+ total);

                } catch (ArithmeticException e) {
                        System.out.println("Exception Occures : " + e.getClass());
                        System.out.println("Exception Message : " + e.getMessage());
                }
        }

        private static void strLengthCalculator(String str) {
                try {
```

```
                int length = str.length();
                System.out.println("Result str lenght :: "+length);
        } catch (NullPointerException e) {
                System.out.println("Exception Occures : " + e.getClass());
                System.out.println("Exception Message : " + e.getMessage());
        }
}

private static void multiConvertor(String input, int number) {
        try {
                int result1 = Integer.parseInt(input);
                int result2 = 2000 / number;
                int result3 = input.length();
                System.out.println("Result1 :> "+result1);
                System.out.println("Result2 :> "+result2);
                System.out.println("Result3 :> "+result3);
        } catch (NumberFormatException e) {
                System.out.println("Exception Occures : " + e.getClass());
                System.out.println("Exception Message : " + e.getMessage());
        } catch (NullPointerException e) {
                System.out.println("Exception Occures : " + e.getClass());
                System.out.println("Exception Message : " + e.getMessage());
        } catch (ArithmeticException e) {
                System.out.println("Exception Occures : " + e.getClass());
                System.out.println("Exception Message : " + e.getMessage());
        } catch (Exception e) {
                System.out.println("Exception Occures : " + e.getClass());
                System.out.println("Exception Message : " + e.getMessage());
        }
}
```

}

13.Finally Block package

```
com.simplilearn.exceptionhandling;  public
class ExceptionHandlingDemo {  public static
void main(String[] args) {
        System.out.println("--- Program started ! ---");
        try {
            int accountBalance = 5000;
        int intrestPer = 0 ;

            int totalIntrest = ( accountBalance / intrestPer ) * 100;

            System.out.println("Total Amount "+ totalIntrest);

        } catch (ArithmeticException e) {
            System.out.println("Exception Occured and handled ::  "+e.getClass());
            System.out.println("Exception Message :: "+e.getMessage());
        }  finally {
            System.out.println("Always executing block : cleanup");
        }
        System.out.println("--- Program ended ! ---");
    }
}
```