Documentation for the OOPS concept

This document records the Object-oriented programming concepts for the final assessment.

Question

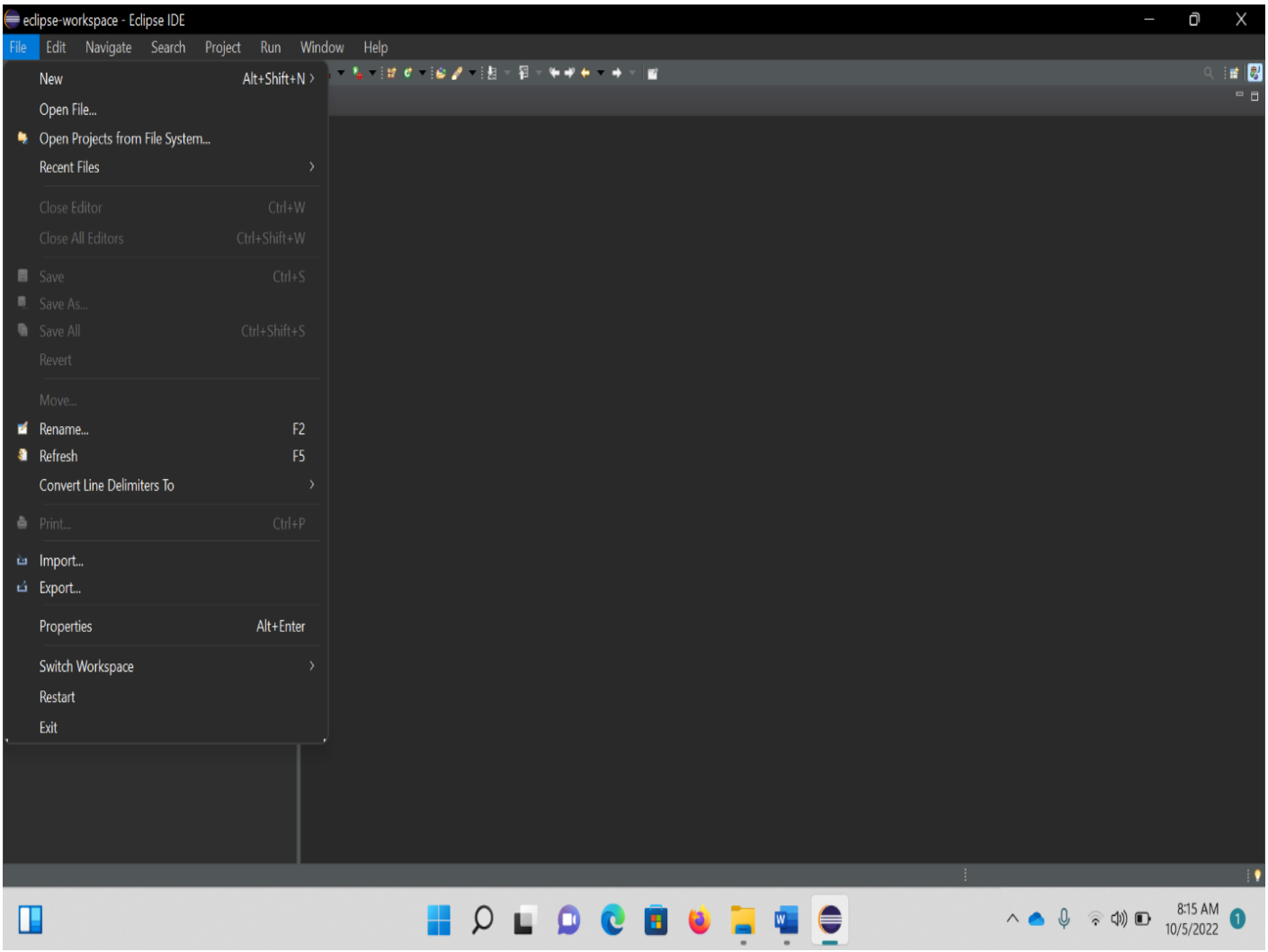
Object Oriented Programming: Write programs on below topics

a) Override
b) Overloading
c) Inheritance
d) Abstract
e) Interface
f) Exception Handling
g) Collections
h) Access Modifiers

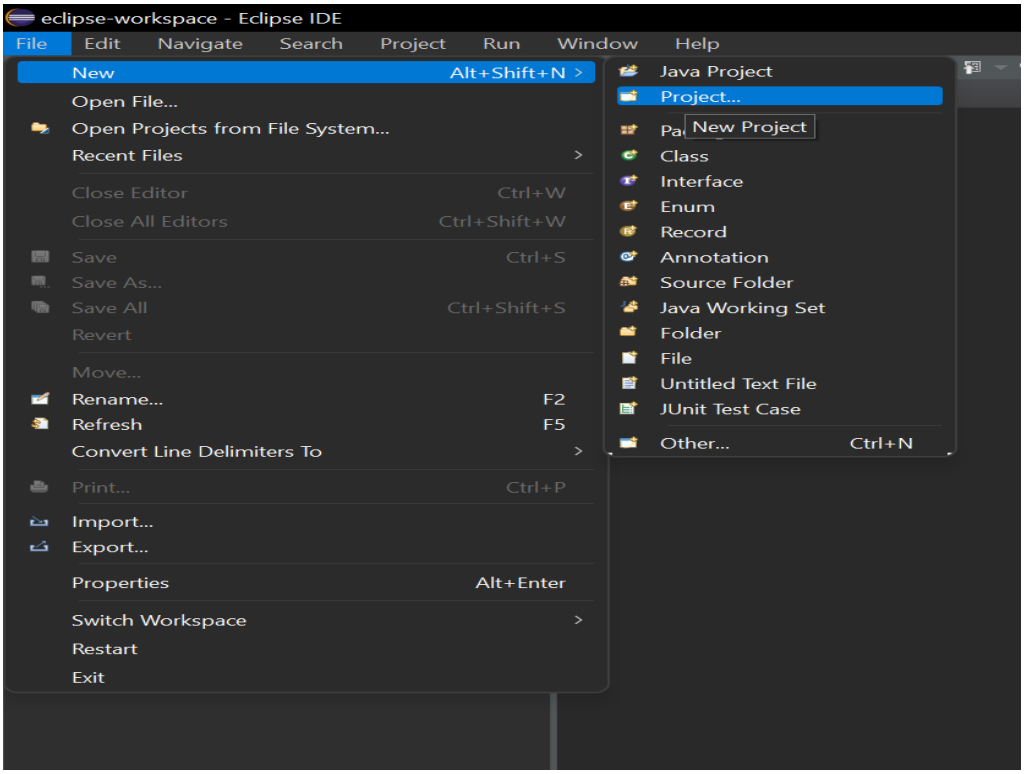
a) Creating a new Java project

Go to file -> New -> Project -> Java project -> enter the name of project

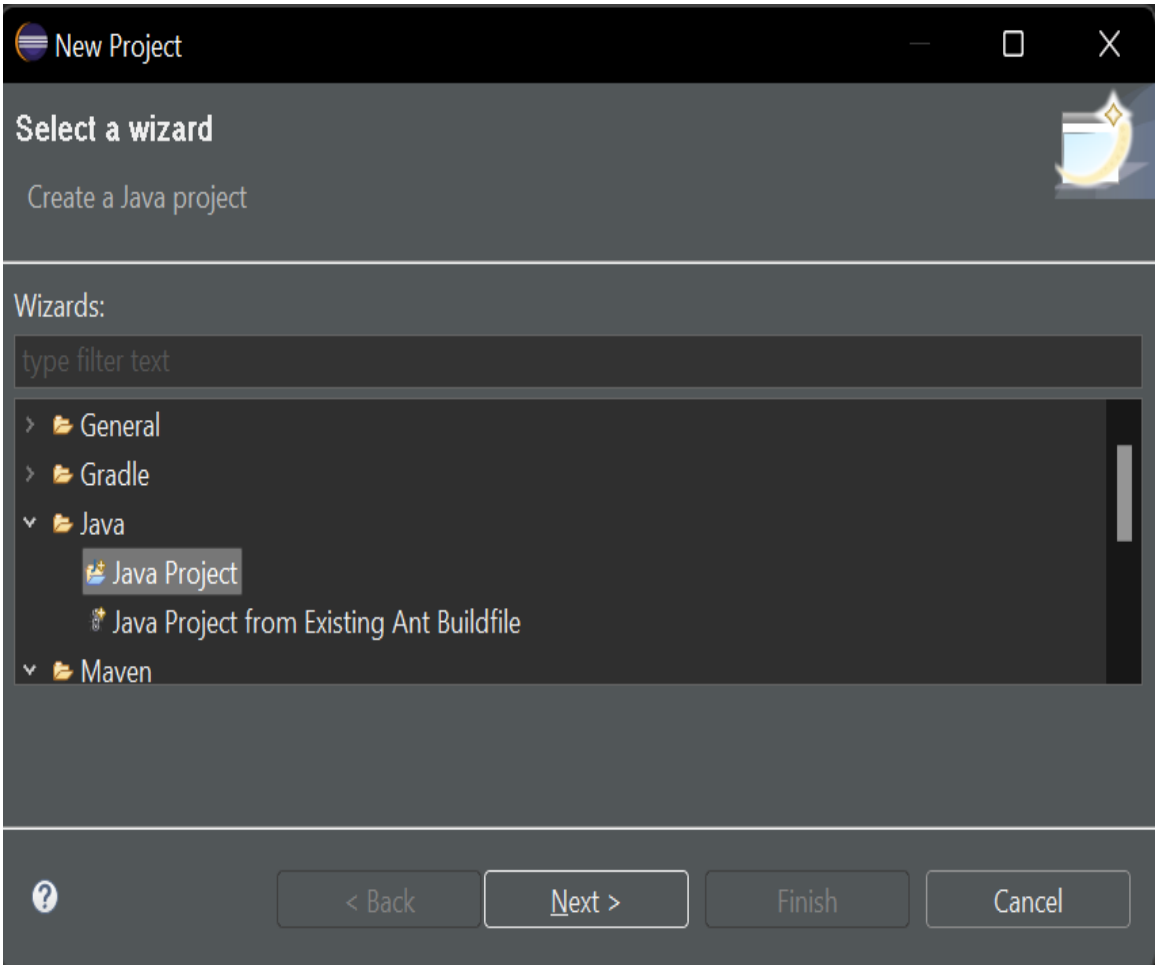
Going to file and then new option



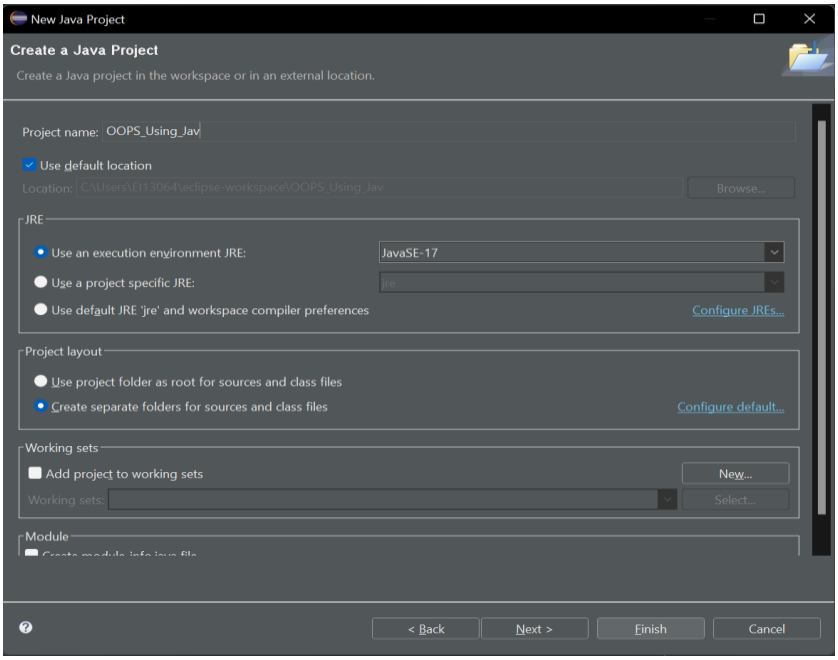
Select the Project option



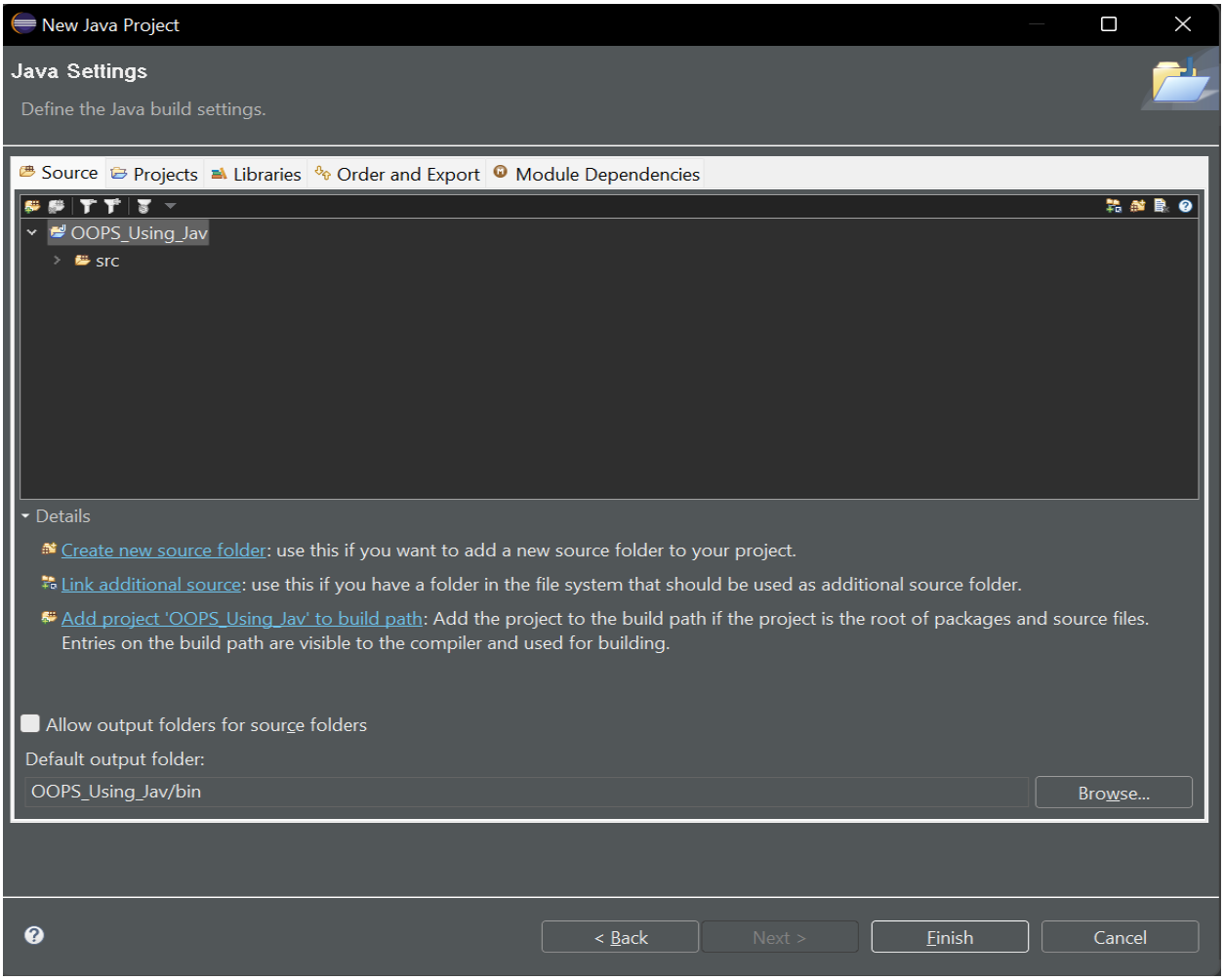
Select Java project



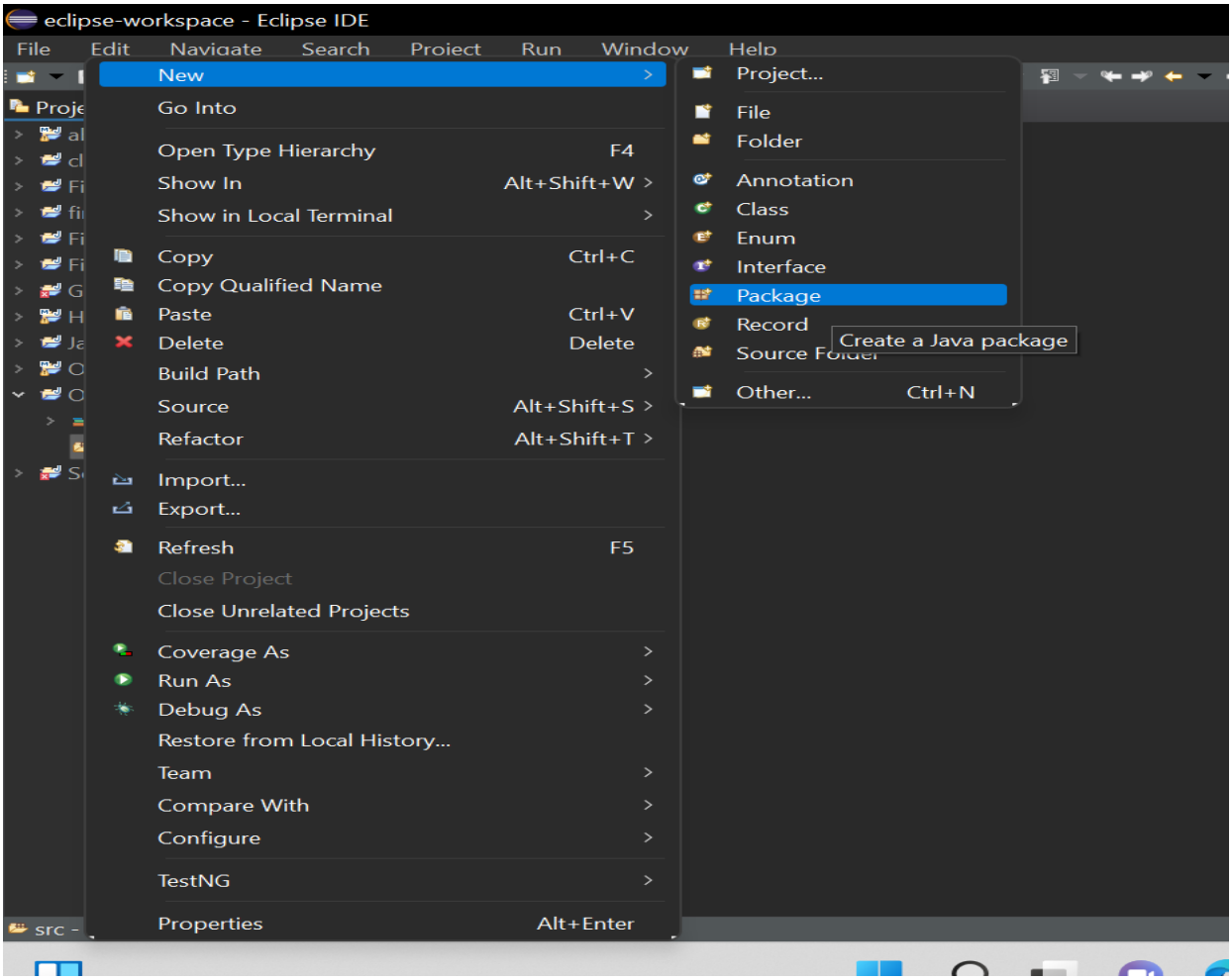
Enter the name of the java project



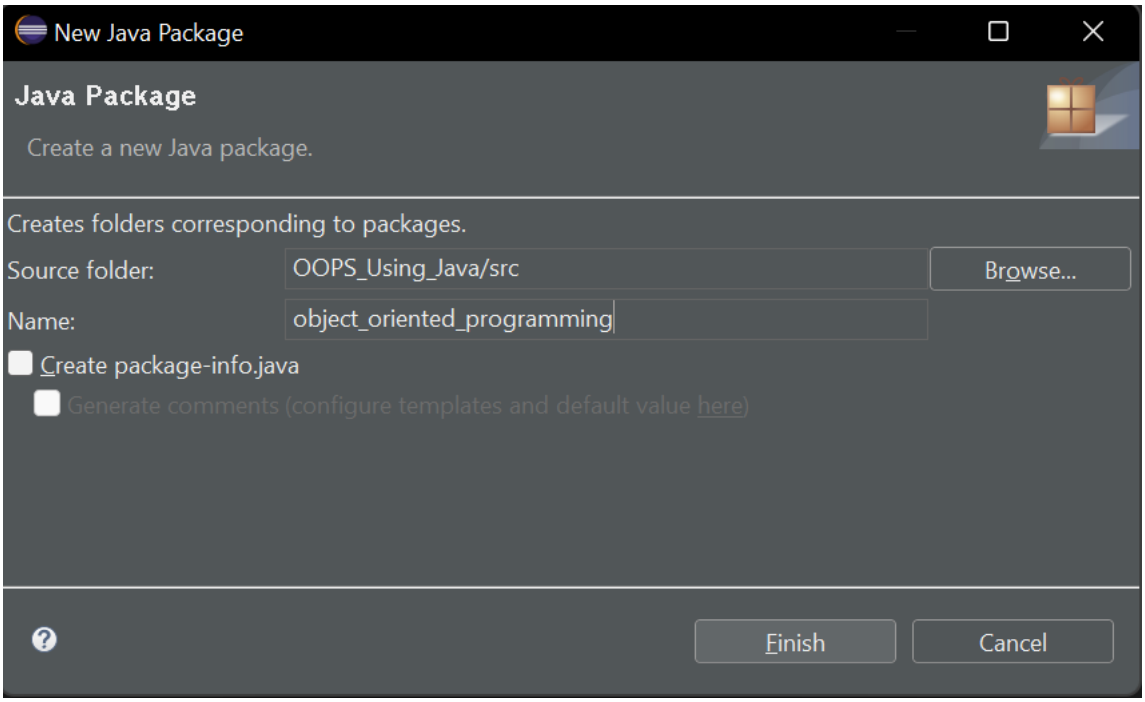
Click on finish



b) Creating a new package inside the Java project



Naming the package



Overriding:

Overriding means a function in the sub class with the same name as that defined in the parent class overrides the one defined in the parent class.
This is done to ensure code reusability, and not having to name different functions

i) In line 5 we create the parent class named Vehicle in which there is a method named manufacturing_details
ii) In line 12, the sub class names bikes inherits the properties of the vehicle parent class, and the class defined in it is the same as that of parent class
iii) In the main function, we create two objects, each one for parent and sub class
iv) In line 23 and 24, we call the same methods but using different objects. Here, the objects used decide which method will be called, because the method names are the same. Therefore overriding happens in line 24.

```
1  package object_oriented_programming;
2
3
4
5      class Vehicle{
6          void manufacturing_details() {
7              System.out.println("150 Vehicles are being manufactured in the showroom");
8          }
9      }
10
11
12     class Bikes extends Vehicle{
13         void manufacturing_details() {
14             System.out.println("Out of total vehicles, 60 are bikes!");
15         }
16     }
17
18     public class Override {
19
20     public static void main(String[] args) {
21           Vehicle obj1 = new Vehicle();
22           Bikes obj2 = new Bikes();
23           obj1.manufacturing_details();
24           obj2.manufacturing_details();
25     }
26
27
```

Overloading:

The most common form of overloading in object-oriented programming is the method overloading.
Method overloading means, when a method behaves in different ways depending upon the number and type of arguments given to it. It's a part of polymorphism

i) In line number 6, 13 and 19, I have defined three methods with the name area. But they work differently based upon the inputs given
ii) One works to calculate the area of square, other for rectangle and the other for circle
iii) In the main function, I create an object of the class and call these methods with appropriate arguments thus performing the method overloading

```java
package object_oriented_programming;

class over_loading_area{

    // This class contains 3 functions named area. Depending upon the parameters, they will perform area calculation of square, rectangle
    void area(int side) {
        System.out.println("Area of the square with side " + side + " is: " + side*side);
    }



    void area(int length, int breadth) {
        System.out.println("Area of the rectangle with length " + length + " and breadth " + breadth + " is: " + 2*(length+breadth) );
    }



    void area(int radius, String cirle) {
        System.out.println("Area of circle with radius " + radius + " is: " + 2*3.14*radius);
    }
}



public class Overloading {

    public static void main(String[] args) {
          over_loading_area obj = new over_loading_area();
          obj.area(12);
          obj.area(2,4);
          obj.area(22,"circle");
    }

}

```

Inheritance

Inheritance is a way to accessing data indirectly through the sub class. It helps to make our code reusable and more readable.

The multilevel inheritance is used in this example.

i) In line 5,10 and 16, I define the grandfather, father, and child class, and they are inheriting each other. Just like the real-life example

ii) The data in the grandfather class is accessible to all the other classes because it is the base class.

iii) The father acts and a subclass to grandfather and a base class to child

iv) In the main function we make an object of sub class father and child and access the details

v) All the details can be accessed from the child class due to multi-level inheritance

```java
package object_oriented_programming;

// This program depicts multilevel inheritance. Here the father class inherits grandfather, and the child class inherits father

class grandfather {
    int grandfather_age  = 78;
    String grandfather_name  = "Albert";
}

class father extends grandfather {
    int father_age = 45;
    String father_name = "Richard";
}

class child extends father {
     int child_age = 22;
     String child_name = "Brayan";
}

public class Inheritance {

    public static void main(String[] args) {
          father obj1 = new father();
          child obj2 = new child();


          System.out.println("Details acessed through subclass - father via inheritance" );
          System.out.println(obj1.father_age);
          System.out.println(obj1.father_name);
          System.out.println(obj1.grandfather_age);
          System.out.println(obj1.grandfather_name);

          System.out.println("Details acessed through subclass - child via inheritance" );
          System.out.println(obj2.child_age);
          System.out.println(obj2.child_name);
          System.out.println(obj2.father_age);
          System.out.println(obj2.father_name);
          System.out.println(obj2.grandfather_age);
          System.out.println(obj2.grandfather_name);
    }
}

```

Abstract:

Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Using abstract classes, the concept of abstraction is used. Also interfaces concept is also the common implementation of abstraction. Interfaces part will be discussed later

Abstract classes are those which can't be instantiated into objects. They can only provide the method definition whereas the implementation is performed in the main class. For this the main class needs to inherit the abstract class.

i) In line 6 I am making an abstract class which has an abstract method named abs. The implementation of this abstract method will be in the main class

ii) In line 10, the main class inherits the abstract class and in line 19 I can implement the abs method.

```
1  package object_oriented_programming;
2
3  // This program explains the concept of abstract classes. An abstract class can't be
4  //instantiated and it's methods can only be implemented in the subclass. Only the method definition will be in abstract class.
5
6  abstract class Abstract_class{
7      abstract void abs();
8  }
9
10 public class Abstract extends Abstract_class{
11     public static void main(String[] args) {
12
13
14         Abstract obj = new Abstract();
15         obj.abs();
16
17     }
18
19     void abs() {
20         System.out.println("Abstract method in a abstract class is defined inside the inherited class");
21     }
22
23 }
24
```

Interfaces:

Interfaces are the most common implementation of abstraction in Object oriented programming

It's also the only way to implement multiple inheritance. This means the interface can inherit multiple interfaces at the same time.

i) In line 6 and 11, I create two interfaces named inter_1 and inter_2.

ii) In line 15, the main class can inherit both interfaces at the same time. This is the multiple inheritance

iii) Since interfaces are a concept of abstraction, the methods in interfaces are implemented in the main class and not in the interface

```java
package object_oriented_programming;

// This program demonstrated the concept of interfaces , used to achieve abstractions in programs. Multiple inheritance in Java
//is possible due to interfaces

interface inter_1{
    void inter_1_method();
}


interface inter_2{
    void inter_2_method();
}
public class Interface implements inter_1,inter_2 {

    public static void main(String[] args) {
           Interface obj = new Interface();
           obj.inter_1_method();
           obj.inter_2_method();
    }


    public void inter_1_method() {
        System.out.println("This abstract method in interface inter_1 is implemented in the sub class");
    }



    public void inter_2_method() {
        System.out.println("This abstract method in interface inter_2 is implemented in the sub class");
    }

}

```

Exception Handling (Try and Catch)

Exception is an unwanted or unexpected event, which occurs during the execution of a program, i.e. at run time, that disrupts the normal flow of the program's instructions. Exception handling is used to gracefully handle those interruptions

In the below example I use the arithmetic exception and index out of bounds exception in line 12 and 15 respectively

i) In line 6, I dynamically declare an array of size 2

ii) In line 10, we two exceptions occur inside the try block. I try to access index 4 that doesn't exist and I try to divide by zero

iii) The right side of the expression is executed first, and thus we get into the arithmetic exception block in catch which prints the divide by zero error

```java
package object_oriented_programming;

public class Exception_Handling {

    public static void main(String[] args) {
          int arr[] = new int[2]; // Dynamically declaring an array


          try {
          arr[4] = 12/0;
          }
          catch(ArithmeticException t){
              System.out.println("Divide by zero error!!");
          }
          catch(IndexOutOfBoundsException i) {
              System.out.println("Array index is out of bounds");
          }
          finally {
              System.out.println("The try catch block is executed");
          }
    }

}
```

Throw

The throw keyword is used to explicitly throw an exception from a method or any block of code.

i) In line 12, I throw an arithmetic exception called “you are underage”. This leads to an error message being printed in the console

ii) In line 15, a throw an index out of bounds exception which leads to an error message in console that the index is out of bounds

```
1  package object_oriented_programming;
2
3  public class Throws {
4
5      public static void main(String[] args) {
6          int age = 10;
7          int size  = 12;
8          int arr[] = new int[size];
9          arr[0] = 12;
10         int k = 11;
11         if(age<18) {
12             throw new ArithmeticException("You are under age");
13         }
14         else if(k>size) {
15             throw new IndexOutOfBoundsException ("Index is out of bounds");
16
17         }
18         else {
19             System.out.println("Age and indexes are valid");
20         }
21
22     }
23
24 }
25
```

Collections

The Java collections framework is a set of classes and interfaces that implement commonly reusable collection data structures.

In the below examples, I have taken three main data structures present in java collection. They are as follows

i) **Array list -** dynamic array for storing the elements. It is like an array, but there is no size limit. We can add or remove elements anytime. So, it is much more flexible than the traditional array.

ii) **Linked list -** Linked List is a part of the Collection framework present in java.util package. This class is an implementation of the LinkedList data structure which is a linear data structure where the elements are not stored in contiguous locations and every element is a separate object with a data part and address part. The elements are linked using pointers and addresses. Each element is known as a node.

iii) **Vector** - Vector implements a dynamic array which means it can grow or shrink as required. Like an array, it contains components that can be accessed using an integer index. They are very similar to ArrayList, but Vector is synchronized and has some legacy methods that the collection framework does not contain.

Array list

In this program, I'm using the utilities package in java to access the Arraylist class

i) In line 10, I declare a new Array list named 'i' which can hold objects of different data type
ii) In lines 12-14, I'm adding elements to the arraylist
iii) Line 16, I am making an iterator pointing to the first element, and later in the while loop, we are printing the elements by accessing it using the iterator
iv) In line 25, I am removing the element present at index 1
v) I am then using the special for each loop to iterate through the array list named i and print the element

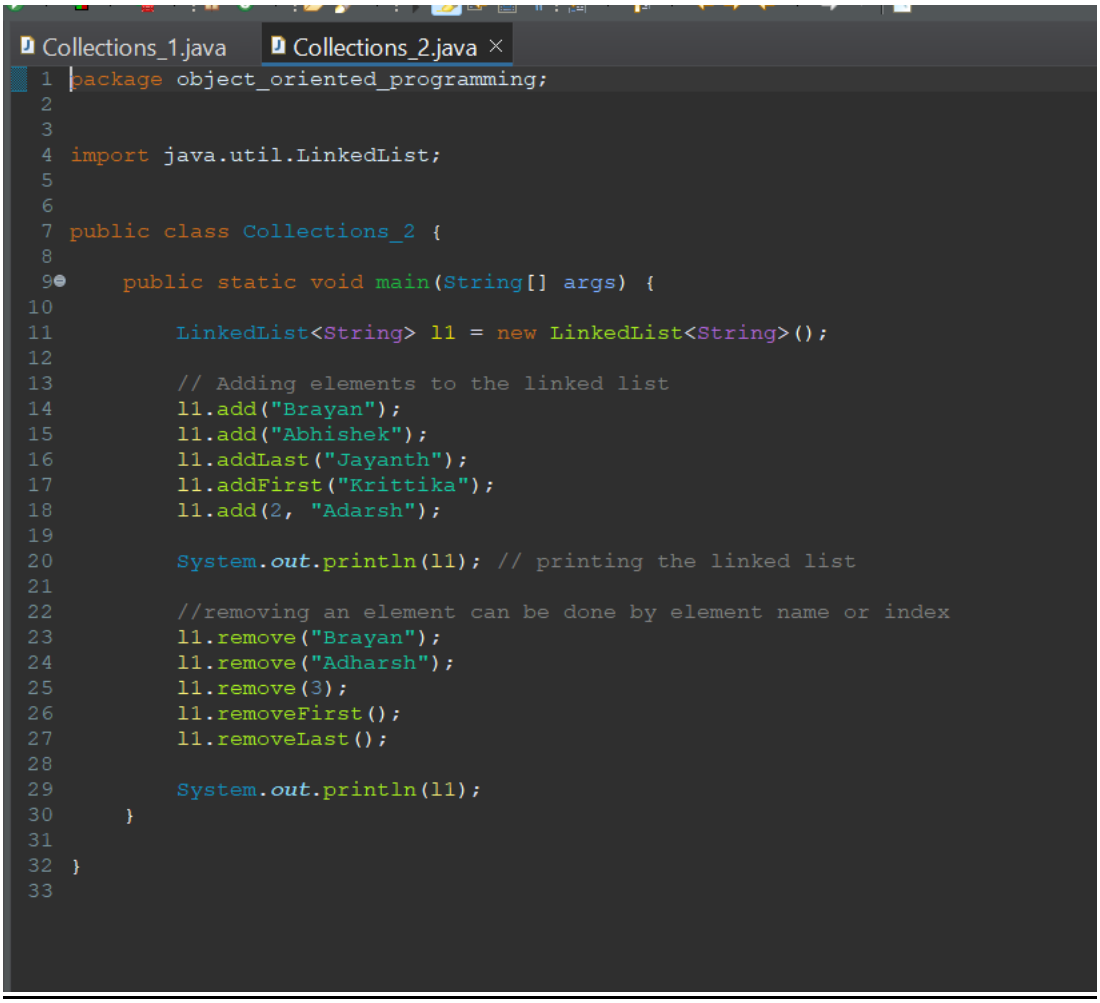
Collections_1.java

```
1  package object_oriented_programming;
2
3  import java.util.ArrayList;
5
6  public class Collections_1 {
7
8      public static void main(String[] args) {
9
10         ArrayList<Object> list = new ArrayList<Object>(); // creating an array list of different data types
11
12         list.add("Brayan"); // adding elements
13         list.add(23);
14         list.add(99);
15
16         Iterator<Object> i = list.iterator();
17
18         // using hasNext method
19         while(i.hasNext())
20         {
21             System.out.println(i.next());
22
23         }
24
25         list.remove(1);
26
27         for(Object i1 : list)
28         {
29             System.out.println(i1);
30
31         }
32
33
34         list.add(2);
35         for(Object i1 : list)
36         {
37             System.out.println(i1);
38
39         }
40
41     }
42 }
43
```

Linked list

In this program, I'm using the utilities package in java to access the linked list class

i) In line 11, I am making an object of linked list class which can hold Strings named l1
ii) From line 14 to 18, I am adding elements to the linked list
iii) In line 20, I am printing the linked list
iv) In line 23-27, I am removing the element from the linked list. It can be removed by index or by element.



Vector

In this program, I'm using the utilities package in java to access the vector class.

i) In line 11 vector object v2 of type integer is created
ii) Using a for loop we can access the elements and populate this vector as shows in lines 15, 16.
iii) In line 19, we print out the vector
iv) In line 22, using the remove method, we can remove the element by means of its index
v) We can use a for loop to access the elements using the get method.
vi) Size method returns the size of the vector i.e., the number of elements in the vector

Collections_1.java Collections_2.java Collections_3.java

```java
package object_oriented_programming;

import java.util.*;

public class Collections_3 {

    public static void main(String[] args) {


        // Declaring the Vector
        Vector<Integer> v2 = new Vector<Integer>();

        // Appending new elements at
        // the end of the list
        for (int i = 1; i <= 5; i++)
            v2.add(i);

        // Printing elements
        System.out.println(v2);

        // Remove element at index 3
        v2.remove(3);

        // Displaying the Vector
        // after deletion
        System.out.println(v2);

        // Printing elements one by one
        for (int i = 0; i < v2.size(); i++)
            System.out.print(v2.get(i) + " ");

    }

}
```

Access Modifiers

The access modifiers in 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 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.

In this example, we have 2 packages namely the object_oriented_programming and the access_specify. These have been formed to show the differences between all the access modifiers

i) In line 4, I declare a private variable that can be accessed only in the access modify class. It can be accessed only at line 14, i.e., inside the class. Line 20 is commented because it lies outside the class and the private variable can't be accessed there or from the main class

ii) In line 5 I declare a protected variable. This can be accessed inside the class as well as outside as shown in 16 and in line 33. It can also be accessed from line 17 of another package in the second snapshot because the main class inherits the parent class

iii) In line 7 I declare a default variable. If there is no access specifier specified then it's a default variable. It can be accessed anywhere inside the package as shown in line 23 and 35

iv) In line 6 I declare the public variable. It can be accessed anywhere inside or outside the package and has the largest scope. It can be accessed outside the package as shown in line 13 of the second snapshot without inheritance of the base class too.

Access_Modifiers.java × Access.java

```
1  package object_oriented_programming;
2
3  class Access_Modify{
4      private int pri_vate = 22;//private
5      protected int pro_tected = 44;//protected
6      public int pub_lic = 88;//public
7      int de_fault = 100;//default
8
9      void print() {
10         System.out.println("Accessing the private variable from the same class: " + pri_vate);
11         //Private data can be accessed only within the base class
12     }}
13
14 class Access_mod extends Access_Modify{
15     void print() {
16         System.out.println("Accessing the protected data from the sub class " + pro_tected);
17         //Protected data can be accessed within the package and outside it only using inheritance
18
19
20         //System.out.println("Accessing the private data from the sub class " + pri_vate) not possible
21
22
23         System.out.println("Accessing the default data from the sub class " + de_fault);// Can be accessed in only the same package
24
25     }}
26
27 public class Access_Modifiers extends Access_mod{
28
29     public static void main(String[] args) {
30         Access_Modify obj1 = new Access_Modify();
31         //System.out.println("Accessing private data by using object of the same class in other class " + obj1.pri_vate);
32         Access_mod obj2 = new Access_mod();
33         System.out.println("Accessing protected data by using object of the sub class in other class " + obj2.pro_tected);
34         System.out.println("Accessing public data from other class " + obj1.pub_lic);
35         System.out.println("Accessing default data: " + obj2.de_fault);
36         //System.out.println("Accessing the private variable from the same class: " + obj1.pri_vate); private data can be accessed only
37         // from the class where it was defined. not outside it
38         }}
39
```

Access_Modifiers.java Access.java

```java
package Access_specify;

import object_oriented_programming.*;


public class Access extends Access_Modifiers {

    public static void main(String[] args) {
        Access obj  = new Access();

        Access_Modifiers obj1 = new Access_Modifiers();

        System.out.println("Acessing the public data from another package: " + obj1.pub_lic);// Accessing public data is possible
        //with or without inheritance.


        System.out.println("Acessing the protected data from another package: " + obj.pro_tected);// This is working because we use
        //inheritance to inherit the base class in this package in line number 6. Without inheritance it won't work



        //System.out.println("Acessing the default data from another package: " + obj.de_fault); will lead to compilation error because
        // the default data can be accessed only in the same package where it is defined.


    }

}
```