Vidyavardhini's

College of Engineering & Technology

Vasai Road (W)

Department of Artificial Intelligence & Data Science

Laboratory Manual Student Copy

Semester	III	Class	S.E.
Course Code	CSL304		
Course Name	Skill based Lab Course: Object Oriented Programming with Java		

Vidyavardhini's College of Engineering& Technology

Vision

To be a premier institution of technical education; always aiming at becoming a valuable resource for industry and society.

Mission

- To provide technologically inspiring environmentfor learning.
- To promote creativity, innovation and professional activities.
- To inculcate ethical and moral values.
- To cater personal, professional and societal needs through quality education.

Department Vision:

To foster proficient artificial intelligence and data science professionals, making remarkable contributions to industry and society.

Department Mission:

- To encourage innovation and creativity with rational thinking for solvingthe challenges in emerging areas.
- To inculcate standard industrial practices and security norms whiledealing with Data.
- To develop sustainable Artificial Intelligence systems for the benefit ofvarious sectors.

Program Specific Outcomes (PSOs):

PSO1: Analyze the current trends in the field of Artificial Intelligence & Data Science and convey their findings by presenting / publishing at a national / international forum.

PSO2: Design and develop Artificial Intelligence & Data Science based solutions and applications for the problems in the different domains catering to industry and society.

Program Outcomes (POs):

Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
- **PO2. Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
- **PO3. Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
- **PO4. Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
- **PO5. Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
- **PO6. The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
- **PO7. Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and teamwork:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
- **PO11. Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
- **PO12. Life-long learning:** Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Objective

1	To learn the basic concept of object-oriented programming
2	To study JAVA Programming language
3	To study various concepts of JAVA programming like multithreading, exception handling, packages etc.
4	To explain components of GUI based application.

Course Outcomes

CO	At the end of course students will be able to:	**Action verbs**	**Bloom's Level**
CSL304.1	Apply the Object Oriented Programming and basic programming constructs for solving problems using JAVA.	Apply	Apply (level 3)
CSL304.2	Apply the concept of packages, classes , objects and accept the input using Scanner and Buffered Reader Class.	Apply	Apply (level 3)
CSL304.3	Apply the concept of strings, arrays, and vectors to perform various operations on sequential data.	Apply	Apply (level 3)
CSL304.4	Apply the concept of inheritance as method overriding and interfaces for multiple inheritance.	Apply	Apply (level 3)
CSL304.5	Apply the concept of exception handling using try, catch, finally, throw and throws and multithreading for thread management.	Apply	Apply (level 3)
CSL304.6	Develop GUI based application using applets and AWT Controls.	Develop	Create (level 6)

Mapping of Experiments with Course Outcomes

List of Experiments	Course Outcomes					
	CSL304.1	CSL304.2	CSL304.3	CSL304.4	CSL304.5	CSL304.6
Implement a program using Basic programming constructs like branching and looping	3	-	-	-	-	-
Implement a program to accept the input from user using Scanner and Buffered Reader.	3	-	-	-	-	-
Implement a program that demonstrates the concepts of class and objects	-	3	-	-	-	-
Implement a program on method and constructor overloading.	-	3	-	-	-	-
Implement a program on Packages.	-	-	3	-	-	-
Implement a program on 2D array & strings functions.	-	-	3	-	-	-
Implement a program on single inheritance.	-	-	-	3	-	-
Implement a program on Multiple Inheritance with Interface.	-	-	-	3	-	-
Implement a program on Exception handling.	-	-	-	-	3	-

Implement a program on Multithreading.	-	-	-	-	3	-
Implement a program on Applet or AWT Controls.	-	-	-	-	-	3
Mini Project based on the content of the syllabus (Group of 2-3 students)	-	-	-	-	-	3

INDEX

Sr. No.	Name of Experiment	D.O.P.	D.O.C.	Page No.	Remark
1	Implement a program using Basic programming constructs like branching and looping				
2	Implement a program to accept the input from user using Scanner and Buffered Reader.				
3	Implement a program that demonstrates the concepts of class and objects				
4	Implement a program on method and constructor overloading.				
5	Implement a program on Packages.				
6	Implement a program on 2D array & strings functions.				
7	Implement a program on single inheritance.				
8	Implement a program on Multiple Inheritance with Interface.				
9	Implement a program on Exception Handling.				
10	Implement a program on Multithreading.				
11	Implement a program on Applet or AWT Controls				
12	Mini Project based on the content of the syllabus (Group of 2-3 students)				

D.O.P: Date of performance

D.O.C : Date of correction

Experiment No.1
Basic programming constructs like branching and looping
Date of Performance:
Date of Submission:

Aim :- To apply programming constructs of decision making and looping.

Objective :- To apply basic programming constructs like Branching and Looping for solving arithmetic problems like calculating factorial of a no entered by user at command prompt .

Theory :-

Programming constructs are basic building blocks that can be used to control computer programs. Most programs are built out of a fairly standard set of programming constructs. For example, to write a useful program, we need to be able to store values in variables, test these values against a condition, or loop through a set of instructions a certain number of times. Some of the basic program constructs include decision making and looping.

Decision Making in programming is similar to decision making in real life. In programming also we face some situations where we want a certain block of code to be executed when some condition is fulfilled. A programming language uses control statements to control the flow of execution of program based on certain conditions. These are used to cause the flow of execution to advance and branch based on changes to the state of a program.

- if
- if-else
- nested-if
- if-else-if
- switch-case
- break, continue

These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

A loop is a programming structure that repeats a sequence of instructions until a specific condition is met. Programmers use loops to cycle through values, add sums of numbers, repeat functions, and many other things. ... Two of the most common types of loops are the while loop and the for loop. The different ways of looping in programming languages are

- while

- do-while

- for loop
- Some languages have modified for loops for more convenience eg :- Modified for loop in java.

For and while loop is entry-controlled loops. Do-while is an exit-controlled loop.

Code: -

a) **FOR LOOP**:

```
class Cont
{
public static void main(String args[])
{
int a=5;
int i;
for(i=0;i<a;i++)
{
if(i%2==0)
{
continue;
}
else{
System.out.print(+i+ " is odd number");
}
}
}
}
```

OUTPUT:

```
Microsoft Windows [Version 10.0.22621.2283]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Dnyanesh>CD C:\Users\Dnyanesh\Desktop\java

C:\Users\Dnyanesh\Desktop\java>javac for.java

C:\Users\Dnyanesh\Desktop\java>java for.java
1 is odd number3 is odd number
C:\Users\Dnyanesh\Desktop\java>
```

b) WHILE LOOP

```
class While
{
public static void main(String args[])
{
int i=0;
while (i<11)
{
System.out.println(+i);
i++;
}
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac while.java

C:\Users\Dnyanesh\Desktop\java>java while.java
0
1
2
3
4
5
6
7
8
9
10
```

c) DO WHILE LOOP

```
class Dowhile
{
public static void main(String args[])
{
int i=1;
do
{
System.out.println(i);
i++;
}while(i<10);
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac Dowhile.java

C:\Users\Dnyanesh\Desktop\java>java Dowhile.java
1
2
3
4
5
6
7
8
9

C:\Users\Dnyanesh\Desktop\java>
```

d) IF STATEMENT:

```
class Ifstatement
{
public static void main(String args[])
{
int a=5;
int b=3;
if(a>b)
{System.out.println("a is greater");}
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac ifstatement.java

C:\Users\Dnyanesh\Desktop\java>java ifstatement.java
a is greater
```

e) IF-ELSE STATEMENT:

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

```
int a=1;
int b=3;
if(a>b)
{System.out.println("a is greater");}
else{
System.out.println("b is greater");}
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>java IfElsestatement.java
b is greater
```

f) IF-ELSE LADDER:

```
class Ladder
{
public static void main(String args[])
{
int a=5;
int b=6;
int c=7;
if(a>b && a>c)
{
System.out.println("a is greatest");
}
else if(b>a && b>c)
{System.out.println("b is greatest");
}
else{
System.out.println("c is greatest");}
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac Ladder.java

C:\Users\Dnyanesh\Desktop\java>java Ladder.java
c is greatest
```

g) SWITCH STATEMENT:

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

```
int ch=4;
switch (ch)
{case 1:
System.out.println("Monday");
break ;
case 2:
System.out.println("Tuesday");
break;
case 3:
System.out.println("Wednesday");
break;
case 4:
System.out.println("Thursday");
break;
case 5:
System.out.println("Friday");
break;
case 6:
System.out.println("Saturday");
break;
case 7:
System.out.println("Sunday");
break;
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac switch.java

C:\Users\Dnyanesh\Desktop\java>java switch.java
Thursday
```

h) CONTINUE STATEMENT:

```
class Cont
{
public static void main(String args[])
{
int a=5;
int i;
for(i=0;i<a;i++)
{
```

```
if(i%2==0)
{
continue;
}
else{
System.out.print(+i+ " is odd number");
}
}
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac cont.java

C:\Users\Dnyanesh\Desktop\java>java cont.java
1 is odd number3 is odd number
```

Conclusion:

1) Comment on how branching and looping useful in solving problems.

Branching and looping are fundamental control structures in Java (and many other programming languages) that are essential for solving a wide range of problems. They provide the means to make decisions and repeat actions, making your code more dynamic and adaptable.

Branching (if statements):

Decision Making: If statements allow you to make decisions in your code based on conditions. You can execute different blocks of code depending on whether a condition is true or false.

Looping:

Repetition: Loops (for, while, and do-while) enable you to repeat a block of code multiple times, which is useful for tasks like processing arrays, lists, and performing iterative calculations.

Experiment No.2
Accepting Input Through Keyboard
Date of Performance:
Date of Submission:

Aim: To apply basic programing for accepting input through keyboard.

Objective: To use the facility of java to read data from the keyboard for any program

Theory:

Java brings various Streams with its I/O package that helps the user perform all the Java input-output operations. These streams support all types of objects, data types, characters, files, etc. to fully execute the I/O operations. Input in Java can be with certain methods mentioned below in the article.

Methods to Take Input in Java

There are two ways by which we can take Java input from the user or from a file

1. BufferedReader Class
2. Scanner Class

Using BufferedReader Class for String Input In Java

It is a simple class that is used to read a sequence of characters. It has a simple function that reads a character another read which reads, an array of characters, and a readLine() function which reads a line.

InputStreamReader() is a function that converts the input stream of bytes into a stream of characters so that it can be read as BufferedReader expects a stream of characters. BufferedReader can throw checked Exceptions.

Using Scanner Class for Taking Input in Java

It is an advanced version of BufferedReader which was added in later versions of Java. The scanner can read formatted input. It has different functions for different types of data types.

The scanner is much easier to read as we don't have to write throws as there is no exception thrown by

it.

It was added in later versions of Java

It contains predefined functions to read an Integer, Character, and other data types as well.

Syntax of Scanner class

Scanner scn = new Scanner(System.in);

Code:

A) INPUT THROUGH JAVA.UTIL PACKAGE(SCANNER CLASS)

```
B) import java.util.Scanner;
C) class UserInput
D) {
E) public static void main(String args[])
F) {
G) Scanner s=new Scanner(System.in);
H) System.out.println("What do you hear?");
I)  String str=s.nextLine();
J) System.out.println("I hear the "+ str);
K) }
L) }
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>java UserInput.java
What do you hear?
drums of liberation
I hear the drums of liberation

C:\Users\Dnyanesh\Desktop\java>
```

B) USER INPUT THROUGH BUFFEREDREADER:

```
import java.io.FileReader;
import java.io.BufferedReader;
class ReadProgram
{
   public static void main(String args[])
     {
       char[] array= new char[100];
```

```
        try
        {
         FileReader File = new FileReader("input.txt");
         BufferedReader input = new BufferedReader(File);
         input.read(array);
         System.out.println("Data in the file;");
         System.out.println(array);
         input.close();
        }
        catch(Exception e)
        {
          e.getStackTrace();
        }
      }
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>java file.java
Data in the file;
moshi moshi
orewa monkey d luffy
kaizokou oni orewa naru
```

Conclusion:

1) Comment on how you have used BufferedReader and Scanner Class for accepting user input

In Java, both the BufferedReader and Scanner classes are commonly used for accepting user input from the command line or other input sources. Each of these classes has its own advantages and use cases, and I'll provide some insights into how they can be used for this purpose.

BufferedReader:

BufferedReader is part of the java.io package and is primarily used for reading text from character input streams. It's efficient for reading large amounts of text efficiently.

Scanner:

The Scanner class is part of the java.util package and is a more high-level and user-friendly way to parse and tokenize input. It can be used for both reading from files and user input.

Experiment No. 3
Implement a program that demonstrates the concepts of class and objects
Date of Performance:
Date of Submission:

Aim: Implement a program that demonstrates the concepts of class and objects

Objective: To develop the ability of converting real time entity into objects and create their classes.

Theory:

A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties i.e., members and methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. Modifiers: A class can be public or has default access.
2. class keyword: class keyword is used to create a class.
3. Class name: The name should begin with a initial letter (capitalized by convention).
4. Superclass (if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. Body: The class body surrounded by braces, {}.

An OBJECT is a basic unit of Object-Oriented Programming and represents the real-life entities. A typical Java program creates many objects, which interact by invoking methods. An object consists of:

1. State: It is represented by attributes of an object. It also reflects the properties of an object.
2. Behavior: It is represented by methods of an object. It also reflects the response of an object with other objects.
3. Identity: It gives a unique name to an object and enables one object to interact with other objects.

Code:

EXAMPLE 1:

```
class Student
{
int id;
String name;
public static void main(String args[])
{
Student s1=new Student();
System.out.print(s1.id);
System.out.print(s1.name);

}

}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac stud.java

C:\Users\Dnyanesh\Desktop\java>java stud.java
0null
C:\Users\Dnyanesh\Desktop\java>
```

EXAMPLE 2:

```
class Employee
{
int id=123;
String name="UT";
public static void main(String args[])
{
Employee e1=new Employee();
System.out.print(e1.id);
System.out.print(e1.name);

}

}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac Emp.java

C:\Users\Dnyanesh\Desktop\java>java Emp.java
123UT
```

Conclusion:

1) Comment on how you create a class template and their objects.

Define a Class Template:
Use the class keyword followed by the class name to define a class template.
Inside the class, declare fields (attributes) to represent the state of objects.
Define constructors to initialize the object's state.
Add methods to define the behavior and actions of the objects.

Create Objects from the Class:
To create objects, use the new keyword followed by the class constructor.
Assign the created objects to variables.

Access Fields and Methods:
Use the dot notation to access fields and call methods of the objects.

Experiment No. 4
Implement a program on method and constructor overloading.
Date of Performance:
Date of Submission:

Aim: Implement a program on method and constructor overloading.

Objective: To use concept of method overloading in a java program to create a class with same function name with different number of parameters.

Theory:

Method Overloading is a feature that allows a class to have more than one method having the same name, if their argument lists are different. It is similar to constructor overloading in Java, that allows a class to have more than one constructor having different argument lists.

Example: This example to show how method overloading is done by having different number of parameters for the same method name.

```
Class DisplayOverloading
{
   public void disp(char c)
   {
       System.out.println(c);
   }
   public void disp(char c, int num)
   {
       System.out.println(c + " "+num);
   }
}
Class Sample
{
  Public static void main(String args[])
  {
     DisplayOverloading obj = new DisplayOverloading();
     Obj.disp('a');
     Obj.disp('a',10);
  }
}
```

Output:

A

A 10

Java supports Constructor Overloading in addition to overloading methods. In Java, overloaded constructor is called based on the parameters specified when a new is executed.

Sometimes there is a need of initializing an object in different ways. This can be done using constructor overloading.

For example, the Thread class has 8 types of constructors. If we do not want to specify anything about a thread then we can simply use the default constructor of the Thread class, however, if we need to specify the thread name, then we may call the parameterized constructor of the Thread class with a String args like this:

Thread t= new Thread (" MyThread ");

Code:

1) **CONSTRUCTOR OVERLOADING**

```
class Overload1
{
public static void main(String[] args)
{ System.out.println(Multiplier.mult(2,3));
 System.out.println(Multiplier.mult(2,2,2));
}
}
class Multiplier
{
 static int mult(int a,int b)
{
return a*b;
}
static int mult(int a,int b,int c)
{
return a*b*c;
}
```

```
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>java Overload1.java
6
8
```

2) METHOD OVERLOADING

```
class Overload2
{
public static void main(String args[])
{
System.out.println(Adder.add(2,3));
System.out.println(Adder.add(1.2,1.2));
}
}
class Adder
{
static int add(int a,int b)
{
return a+b;
}
static double add(double a,double b)
{return a+b;}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac Overload2.java

C:\Users\Dnyanesh\Desktop\java>java Overload2.java
5
2.4
```

Conclusion:

Comment on how function and constructor overloading used using java
Function and constructor overloading in Java involves creating multiple methods or constructors with the same name within a class but with different parameter lists.

Function Overloading:
Function overloading allows you to define multiple methods in a class with the same name but different parameter lists (number or types of parameters). This way, you can provide different behavior for the

same operation depending on the inputs. The choice of which method to call is made at compile-time based on the provided arguments.

Constructor Overloading:

Constructor overloading is similar to function overloading but is used to define multiple constructors within a class with different parameter lists. Constructor overloading enables the creation of objects in various ways, depending on the arguments provided during object instantiation. Like function overloading, constructors can have different parameter types, but the number and types of parameters should differ to distinguish between them.

Experiment No. 5
Implement a program on Packages.
Date of Performance:
Date of Submission:

Aim: To use packages in java.

Objective: To use packages in java to use readymade classes available in them using square root method in math class.

Theory:

A java package is a group of similar types of classes, interfaces and sub-packages. Packages are used in Java in order to prevent naming conflicts, to control access, to make searching/locating and usage of classes, interfaces, enumerations and annotations easier, etc.

There are two types of packages-

1. Built-in package: The already defined package like java.io.*, java.lang.* etc are known as built-in packages.
2. User defined package: The package we create for is called user-defined package.

Programmers can define their own packages to bundle group of classes/interfaces, etc. While creating a package, the user should choose a name for the package and include a package statement along with that name at the top of every source file that contains the classes, interfaces, enumerations, and annotation types that you want to include in the package. If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.

Code:

```
package mypack;
class Hello
{
public static void main(String args[])
{
System.out.println("Yo");
}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac hello.java

C:\Users\Dnyanesh\Desktop\java>java hello.java
Yo
```

Conclusion:

Comment on the autoencoder architecture and the Image compression results.
Autoencoders are neural networks used for data compression. They consist of an encoder to reduce data dimensions and a decoder to reconstruct the data. In Java, you can build an autoencoder for image compression. Results will include smaller-sized images that maintain essential features, useful for storage and transmission, but with some loss of detail due to the compression.

Experiment No. 6
Implement a program on 2D array & strings functions.
Date of Performance:
Date of Submission:

Aim: To use 2D arrays and Strings for solving given problem.

Objective: To use 2D array concept and strings in java to solve real world problem

Theory:

- An array is used to store a fixed-size sequential collection of data of the same type.
- An array can be init in two ways:
  1. Initializing at the time of declaration:
     dataType[] myArray = {value0, value1, ..., valuek};
  2. Dynamic declaration:
     dataType[] myArray = new dataType[arraySize];
     myArray[index] = value;
- Two – dimensional array is the simplest form of a multidimensional array. Data of only same data type can be stored in a 2D array.Data in a 2D Array is stored in a tabular manner which can be represented as a matrix.
- A 2D Array can be declared in 2 ways:
  1. Intializing at the time of declaration:
     dataType[][] myArray = { {valueR1C1, valueR1C2...}, {valueR2C1, valueR2C2...},..}
  2. Dynamic declaration:
     dataType[][] myArray = new dataType[x][y];
     myArray[row_index][column_index] = value;

In Java, string is basically an object that represents sequence of char values. An array of characters works same as Java string. **Java String** class provides a lot of methods to perform operations on strings such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

1.String literal

To make Java more memory efficient (because no new objects are created if it exists already in the string constant pool).

Example:

String demoString = "GeeksforGeeks";

2. Using new keyword

- String s = new String("Welcome");
- In such a case, JVM will create a new string object in normal (non-pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in the heap (non-pool)

Example:

String demoString = new String ("GeeksforGeeks");

Code:

1) 1-D ARRAY

```
2) import java.util.Scanner;
3) class Array
4) {
5) public static void main(String[] args)
6)     {
7)     int i, n;
8)     Scanner sc=new Scanner(System.in);
9)     System.out.println("Enter Number of elements in an Array :");
10)    n = sc.nextInt();
11)    int Array[] = new int[n];
12)    System.out.println("Enter the elements  :");
13)    for(i=0;i<n;i++)
14)    {
15)        Array[i]=sc.nextInt();
16)}
17)System.out.println("elements in the array are:");
18)for(i=0;i<n;i++)
19){
20)System.out.println(Array[i]);
21)}
22)}
23)}
```

OUTPUT:-

```
C:\Users\Dnyanesh\Desktop\java>javac Array.java

C:\Users\Dnyanesh\Desktop\java>java Array.java
Enter Number of elements in an Array :
3
Enter the elements  :
6 7 8
elements in the array are:
6
7
8
```

2) 2-D ARRAY

```
import java.util.Scanner;
class Matrix
{
public static void main(String[] args)
    {
    int i,j,r,c;
    Scanner sc=new Scanner(System.in);
    System.out.println("Enter Number of rows :");
    r = sc.nextInt();
        System.out.println("Enter number of columns:");
        c = sc.nextInt();
    int Matrix[][]= new int[r][c];
    System.out.println("Enter the elements  :");
    for(i=0;i<r;i++)
    {
         for(j=0;j<c;j++)
{
        Matrix[i][j]=sc.nextInt();
}
System.out.println(" ");
}

System.out.println("elements in the array are:");
for(i=0;i<r;i++)
{
for(j=0;j<c;j++)
{
System.out.print(Matrix[i][j] + " ");
}
```

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

OUTPUT:-

```
C:\Users\Dnyanesh\Desktop\java>java mATRIX.java
Enter Number of rows :
2
Enter number of columns:
2
Enter the elements  :
2 3 4 5


elements in the array are:
2 3
4 5
```

3. STRINGS

```
class Tp
{
int id;
String name;
Tp(String s,int i)
{id=i;
name=s;
}
public static void main(String args[])
{
Tp tp=new Tp("Dnyanesh",22);
System.out.println("Emp name:\t"+ tp.name+"\tid"+tp.id);
}
}
```

OUTPUT:-

```
C:\Users\Dnyanesh\Desktop\java>javac Tp.java

C:\Users\Dnyanesh\Desktop\java>java Tp.java
Emp name:       Dnyanesh        id22

C:\Users\Dnyanesh\Desktop\java>
```

Conclusion:

Comment on how you have used the concept of string and 2D array.

Use of string: The **Tp** class has a constructor with two parameters, **String s** and **int i**. In this context, the **String s** parameter is used to initialize the **name** attribute of a **Tp** object, and the **int i** parameter is used to initialize the **id** attribute.
In the **main** method, a **Tp** object named **tp** is created. The constructor is called with the string "Dnyanesh" and the integer 22 as arguments. This sets the **name** attribute of the **tp** object to "Dnyanesh" and the **id** attribute to 22.
Finally, a message is printed to the console using **System.out.println()**. The message includes the values of the **name** and **id** attributes of the **tp** object, which are concatenated into the output string. The **name** attribute is accessed using **tp.name**, and the **id** attribute is accessed using **tp.id**. Both attributes are used in the output string to display the employee's name and ID.
Use of 2D array : The program first uses the **Scanner** class to read input from the user. It prompts the user to enter the number of rows and columns for the matrix, and these values are stored in the variables **r** and **c**, respectively.
An integer 2D array is declared and initialized with the dimensions specified by the user's input. The array is defined as **int Matrix[][] = new int[r][c]**, creating a matrix with **r** rows and **c** columns.
The program then prompts the user to enter the elements of the matrix. It uses nested **for** loops to iterate over each cell in the 2D array. The outer loop iterates over rows (from 0 to **r-1**), and the inner loop iterates over columns (from 0 to **c-1**). The user-provided values are read with **sc.nextInt()** and stored in the corresponding cell of the 2D array **Matrix[i][j]**.
After the user enters all the elements, the program displays the elements stored in the 2D array. It uses another set of nested **for** loops to iterate over the entire array, printing each element with **System.out.print()** and separating them with spaces. A newline character is added after each row to format the output.

Experiment No. 7
Implement a program on single inheritance.
Date of Performance:
Date of Submission:

Aim: To implement the concept of single inheritance.

Objective: Ability to design a base and child class relationship to increase reusability.

Theory:

Single inheritance can be defined as a derived class to inherit the basic methods (data members and variables) and behaviour from a superclass. It's a basic is-a relationship concept exists here. Basically, java only uses a single inheritance as a subclass cannot extend more superclass.

Inheritance is the basic properties of object-oriented programming. Inheritance tends to make use of the properties of a class object into another object. Java uses inheritance for the purpose of code-reusability to reduce time by then enhancing reliability and to achieve run time polymorphism. As the codes are reused it makes less development cost and maintenance. Java has different types of inheritance namely single inheritance, multilevel, multiple, hybrid. In this article, we shall go through on basic understanding of single inheritance concept briefly in java with a programming example. Here we shall have a complete implementation in java.

Syntax:

The general syntax for this is given below. The inheritance concepts use the keyword 'extend' to inherit a specific class. Here you will learn how to make use of extending keyword to derive a class. An extend keyword is declared after the class name followed by another class name. Syntax is,

```
class base class
{…. methods
}
class derived class name extends base class
{
methods … along with this additional feature
}
```

Java uses a keyword 'extends' to make a new class that is derived from the existing class. The inherited

class is termed as a base class or superclass, and the newly created class is called derived or subclass. The class which gives data members and methods known as the base class and the class which takes the methods is known as child class.

Code:

```
class example extends SingleInheritance{
int bonus=10000;
public static void main(String args[]){
example a=new example();
System.out.println("Salary :"+a.salary);
System.out.println("bonus :"+a.bonus);
}
}
class SingleInheritance{
float salary=40000;
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac SingleInheritance.java

C:\Users\Dnyanesh\Desktop\java>java SingleInheritance.java
Salary :40000.0
bonus :10000
```

Conclusion:

Comment on the Single inheritance.

Single inheritance in Java refers to the concept where a class can inherit the properties and behaviors of only one superclass. In other words, a Java class can have at most one direct parent class. This is a key aspect of Java's class inheritance hierarchy. In a single inheritance scenario, a Java class (subclass or derived class) can extend only one other class (superclass or base class). This means that it can inherit the fields and methods of that specific superclass.

Experiment No. 8
Implement a program on multiple inheritance with interface.
Date of Performance:
Date of Submission:

Aim: Implement a program on multiple inheritance with interface.

Objective: Implement multiple inheritance in a program to perform addition, multiplication and transpose operations on a matrix. Create an interface to hold prototypes of these methods and create a class input to read input. Inherit a new class from this interface and class. In main class create object of this child class and invoke required methods.

Theory:

- In Multiple inheritance, one class can have more than one superclass and inherit features from all parent classes. Java does not support multiple inheritance with classes. In java, we can achieve multiple inheritance only through Interfaces.
- An interface contains variables and methods like a class but the methods in an interface are abstract by default unlike a class. If a class implements multiple interfaces, or an interface extends multiple interfaces, it is known as multiple inheritance.
- However, Java supports multiple interface inheritance where an interface extends more than one super interfaces.
- A class implements an interface, but one interface extends another interface. Multiple inheritance by interface occurs if a class implements multiple interfaces or also if an interface itself extends multiple interfaces.
- The following is the syntax used to extend multiple interfaces in Java:

```
access_specifier interface subinterfaceName extends superinterface1, superinterface2, …… {
// Body
}
```

Code:

```
class MultipleInheritance{
public static void main(String args[])
{
Pig a=new Pig();
a.animalsound();
```

```
a.sleep();
}
}
interface Animal{
public void animalsound();
public void sleep();
}
class Pig implements Animal{
public void animalsound(){
System.out.println("The pig says: oink oink");
}
public void sleep(){
System.out.println("zzzzzzzzzzzzz");}
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac MultipleInheritance.java

C:\Users\Dnyanesh\Desktop\java>java MultipleInheritance.java
The pig says: oink oink
zzzzzzzzzzzzz
```

Conclusion:

Comment on how interface are useful and implemented using java.

Interfaces in Java are a fundamental concept that allows you to define a contract specifying a set of methods that implementing classes must adhere to.

Abstraction: Interfaces allow you to define a contract or a set of methods without specifying the implementation. This promotes abstraction, enabling you to focus on what a class should do rather than how it should do it.

Experiment No. 9
Implement a program on Exception handling.
Date of Performance:
Date of Submission:

Aim: Implement a program on Exception handling.

Objective: To able handle exceptions occurred and handle them using appropriate keyword

Theory:

The Exception Handling in Java is one of the powerful mechanisms to handle the runtime errors so that the normal flow of the application can be maintained.

Exception Handling is a mechanism to handle runtime errors such as ClassNotFoundException, IOException, SQLException, RemoteException, etc.

Java Exception Keywords

Java provides five keywords that are used to handle the exception. The following table describes each.

Keyword	Description
try	The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally.
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 necessary 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 specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature.

```
public class JavaExceptionExample{

 public static void main(String args[]){

  try{

    //code that may raise exception

    int data=100/0;
```

```
}catch(ArithmeticException e){System.out.println(e);}

 //rest code of the program

 System.out.println("rest of the code...");

}

}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
rest of the code...
```

Code:

1} Try-catch

```
class Main2
{
public static void main(String args[])
{
try{
  int divideByZero = 8/0;
  System.out.println("Rest of code in try block");
    }

   catch (ArithmeticException e) {
     System.out.println("ArithmeticException => " + e.getMessage());
   }
  }
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>javac Main2.java

C:\Users\Dnyanesh\Desktop\java>java Main2.java
ArithmeticException => / by zero
```

2} finally

```
class TestFinallyBlock {
  public static void main(String args[]){
  try{
   int data=25/5;
   System.out.println(data);
```

```
  }
  catch(NullPointerException e){
System.out.println(e);
}
 finally {
System.out.println("finally block is always executed");
}

System.out.println("rest of phe code...");
  }
}
```

OUTPUT:

```
C:\Users\Dnyanesh\Desktop\java>java TestFinallyBlock.java
5
finally block is always executed
rest of the code...
```

3}throws

```
import java.io.IOException;
 class Testthrows2{
   public static void main(String args[]){
    try{
     M m=new M();
     m.method();
    }catch(Exception e){System.out.println("exception handled");}

    System.out.println("normal flow...");
  }
}
class M {
    void method() throws IOException {
        throw new IOException("device error");
    }
}
```

OUTPUT:-

```
C:\Users\Dnyanesh\Desktop\java>javac Testthrows2.java

C:\Users\Dnyanesh\Desktop\java>java Testthrows2.java
exception handled
normal flow...
```

4} throw

```
 class TestThrow3
{
```

```
   public static void main(String args[])
   {
      try
      {
         throw new UserDefinedException("This is user-defined exception");
      }
      catch (UserDefinedException ude)
      {
         System.out.println("Caught the exception");
         System.out.println(ude.getMessage());
      }
   }
}
class UserDefinedException extends Exception
{
   public UserDefinedException(String str)
   {
      super(str);
   }
}
```

OUTPUT:-

```
C:\Users\Dnyanesh\Desktop\java>javac TestThrow3.java

C:\Users\Dnyanesh\Desktop\java>java TestThrow3.java
Caught the exception
This is user-defined exception
```

Conclusion:

Comment on how exceptions are handled in JAVA.

In Java, exceptions are handled using a combination of the try, catch, finally, and throw keywords. Exception handling is a crucial aspect of Java programming, as it allows you to gracefully deal with runtime errors and maintain the stability and reliability of your programs.

Try-Catch Blocks (Using try and catch): The primary mechanism for handling exceptions is the try-catch block. Code that may potentially throw an exception is placed within a try block, and you provide one or more catch blocks to handle specific types of exceptions.

Finally Block (Using finally): You can also use a finally block after the try-catch blocks. Code

within the finally block is executed regardless of whether an exception was thrown or not. It's typically used for cleanup actions (e.g., closing resources).

Throwing Exceptions (Using throw): You can use the throw keyword to explicitly throw an exception within your code. This is often done when you encounter an exceptional situation that your code can't handle, and you want to pass the control to an exception handler.

Experiment No. 10
Implement program on Multithreading
Date of Performance:
Date of Submission:

Aim: Implement program on Multithreading

Objective:

Theory:

Multithreading in Java is a process of executing multiple threads simultaneously.

A thread is a lightweight sub-process, the smallest unit of processing. Multiprocessing and multithreading, both are used to achieve multitasking.

However, we use multithreading than multiprocessing because threads use a shared memory area. They don't allocate separate memory area so saves memory, and context-switching between the threads takes less time than process.

Java Multithreading is mostly used in games, animation, etc.

Java provides **Thread class** to achieve thread programming. Thread class provides constructors and methods to create and perform operations on a thread. Thread class extends Object class and implements Runnable interface.

There are two ways to create a thread:

1. By extending Thread class
2. By implementing Runnable interface.

Thread class:
Thread class provide constructors and methods to create and perform operations on a thread.Thread class extends Object class and implements Runnable interface.

1) Java Thread Example by extending Thread class
FileName: Multi.java

```
class Multi extends Thread{
public void run(){
System.out.println("thread is running...");
}
public static void main(String args[]){
Multi t1=new Multi();
t1.start();
 }
```

```
}
```

Output:

```
thread is running...
```

2) Java Thread Example by implementing Runnable interface

FileName: Multi3.java

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

Output:

```
thread is running...
```

Code:

```
class Multi3 implements Runnable{
public void run(){
System.out.println("thread is running...");
}

public static void main(String args[]){
Multi3 m1=new Multi3();
Thread t1 =new Thread(m1);   // Using the constructor Thread(Runnable r)
t1.start();
 }
}
```

OUTPUT:-

```
C:\Users\Dnyanesh\Desktop\java>java Multi3.java
thread is running...
```

Conclusion:

Comment on how multithreading is supported in JAVA.

Multithreading in Java is supported through the Thread class and the Runnable interface. You can create and manage threads by extending the Thread class or implementing the Runnable interface. Java provides thread synchronization, management, and various thread states to enable concurrent execution of tasks. It's a fundamental feature for efficient resource utilization, improved application responsiveness, and better performance in multi-tasking environments.

Experiment No. 11
Implement a program on Applet or AWT Controls
Date of Performance:
Date of Submission:

Aim: Implement a program on Applet or AWT Controls

Objective:

To develop application like Calculator, Games, Animation using AWT Controls.

Theory:

Java AWT (Abstract Window Toolkit) is an API to develop Graphical User Interface (GUI) or windows-based applications in Java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavy weight i.e. its components are using the resources of underlying operating system (OS).

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

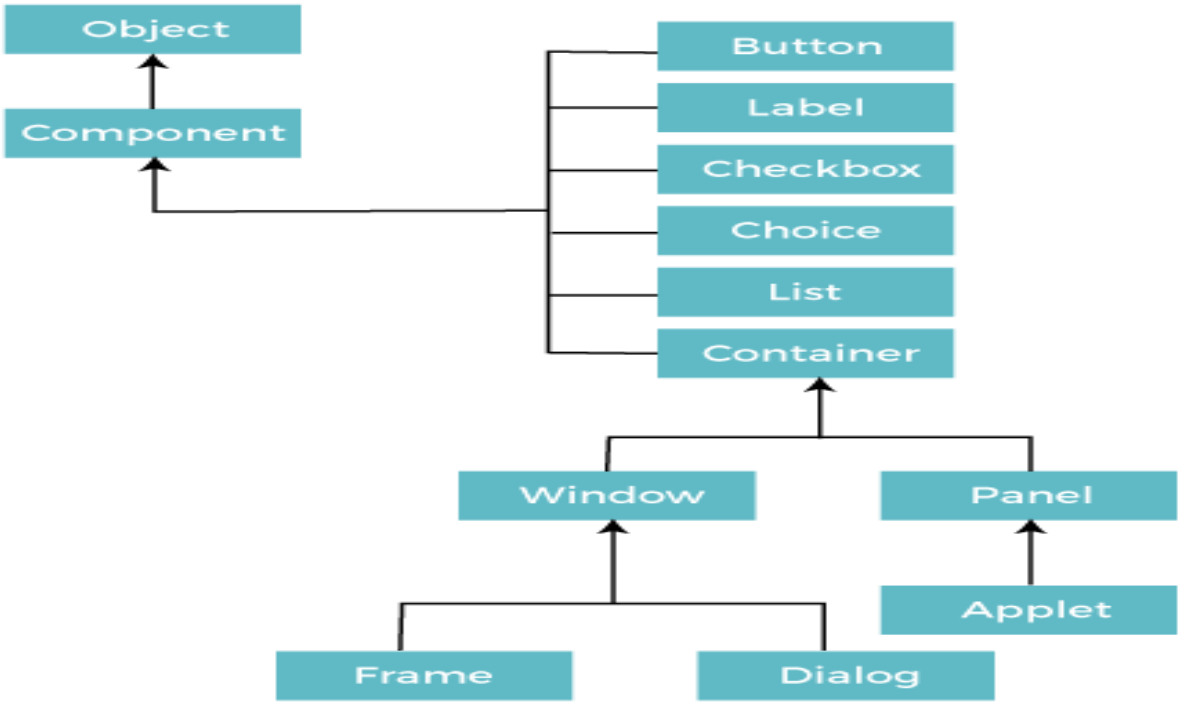
1. A general interface between Java and the native system, used for windowing, events and layout managers. This API is at the core of Java GUI programming and is also used by Swing and Java 2D. It contains the interface between the native windowing system and the Java application1.
2. A basic set of GUI widgets such as buttons, text boxes, and menus1. AWT also provides Graphics and imaging tools, such as shape, color, and font classes2. AWT also avails layout managers which helps in increasing the flexibility of the window layouts2

Java AWT calls the native platform calls the native platform (operating systems) subroutine for creating API components like TextField, ChechBox, button, etc.

For example, an AWT GUI with components like TextField, label and button will have different look and feel for the different platforms like Windows, MAC OS, and Unix. The reason for this is the platforms have different view for their native components and AWT directly calls the native subroutine that creates those components.

In simple words, an AWT application will look like a windows application in Windows OS whereas it will look like a Mac application in the MAC OS.

Java AWT Hierarchy

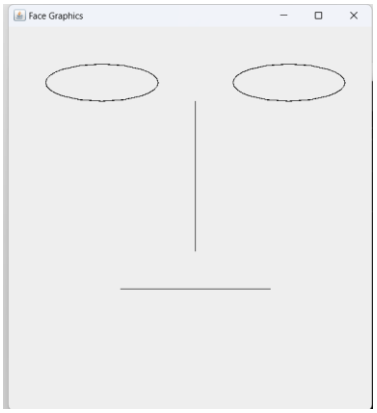


Code:

```
import javax.swing.*;
import java.awt.*;

class Face extends JPanel {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Drawing shapes

        g.setColor(Color.BLACK);
        g.drawOval(50, 50, 150, 50);

        g.setColor(Color.BLACK);
        g.drawOval(300, 50, 150, 50);
```

```
        g.setColor(Color.BLACK);
        g.drawLine(250, 100, 250, 300);

        g.setColor(Color.BLACK);
        g.drawLine(150, 350, 350, 350);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            JFrame frame = new JFrame("Face Graphics");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.add(new Face());
            frame.setSize(500, 550);
            frame.setVisible(true);
        });
    }
}
```

OUTPUT:-



Conclusion:

Comment on application development using AWT Controls.

Application development using AWT (Abstract Window Toolkit) controls in Java involves creating graphical user interfaces (GUIs) for desktop applications. AWT provides a set of basic GUI components, such as buttons, labels, text fields, and more. Here's a brief overview:

1. AWT Controls: AWT offers GUI controls for building your application's user interface.
2. Layout Managers: AWT provides layout managers to arrange and position controls within

your GUI.

3. Customization: You can customize the appearance and behavior of AWT controls.

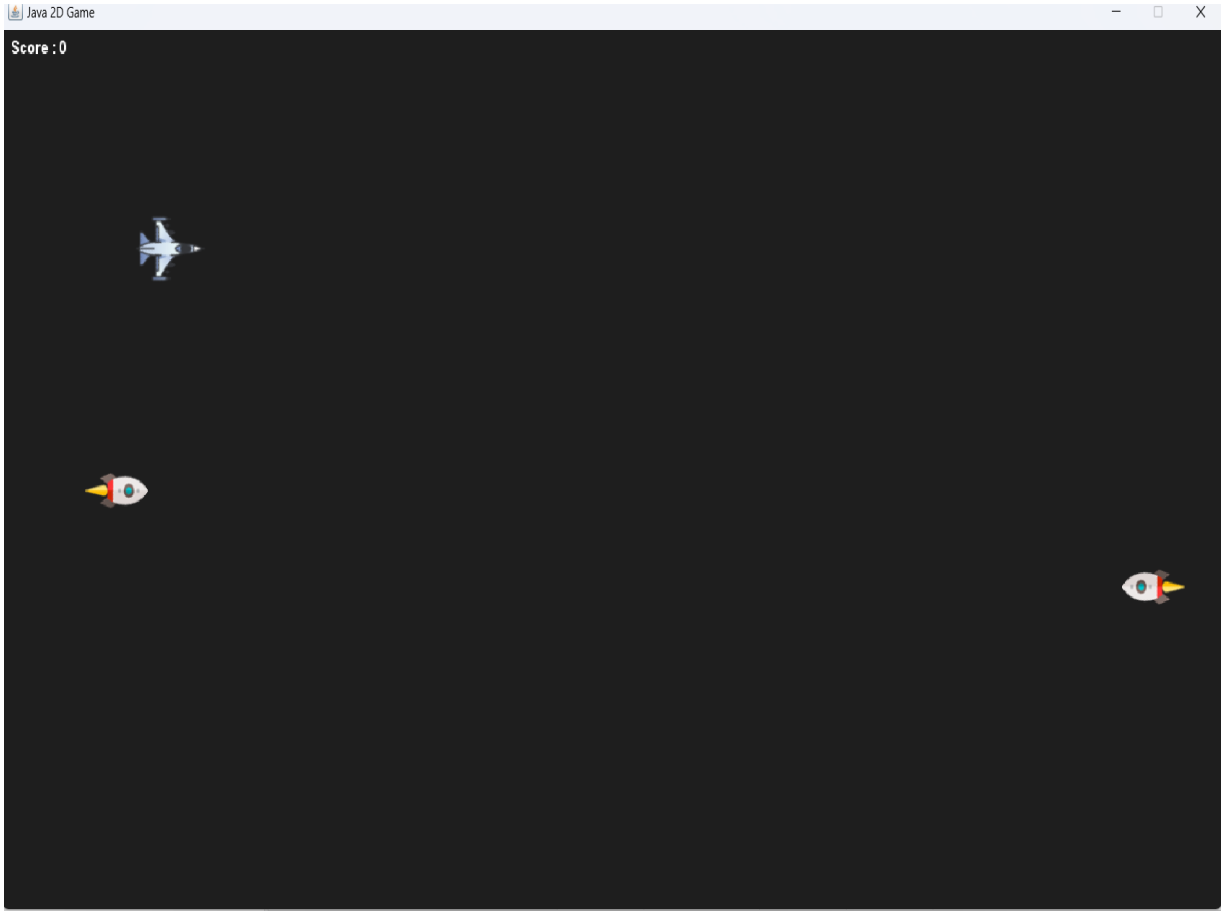
4. Platform Independence: AWT is platform-independent but may not provide the most modern look and feel.

5. Window and Frame: AWT allows you to create top-level containers (e.g., `Frame`) as the main windows for your application.

Experiment No. 12
Course Project based on the content of the syllabus.
Date of Performance:
Date of Submission:

SPACE RUSH-THE GAME

OUTPUT:-



PROGRAM:-

```
package game.main;

import game.component.PanelGame;
import java.awt.BorderLayout;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class Main extends Jframe {

    public Main() {
        init();
    }

    private void init() {
        setTitle("Java 2D Game");
        setSize(1366, 768);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(Jframe.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());
        PanelGame panelGame = new PanelGame();
        add(panelGame);
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowOpened(WindowEvent e) {
                panelGame.start();
            }
        });
    }

    public static void main(String[] args) {
        Main main = new Main();
```

```
package game.component;

public class Key {

    public 64oolean isKey_enter() {
        return key_enter;
    }

    public void setKey_enter(64oolean key_enter) {
        this.key_enter = key_enter;
    }

    public 64oolean isKey_right() {
        return key_right;
    }

    public void setKey_right(64oolean key_right) {
        this.key_right = key_right;
    }

    public 64oolean isKey_left() {
        return key_left;
    }

    public void setKey_left(64oolean key_left) {
        this.key_left = key_left;
    }

    public 64oolean isKey_space() {
        return key_space;
    }

    public void setKey_space(64oolean key_space) {
        this.key_space = key_space;
    }

    public 64oolean isKey_j() {
        return key_j;
    }

    public void setKey_j(64oolean key_j) {
        this.key_j = key_j;
    }

    public 64oolean isKey_k() {
        return key_k;
```

```
    }

    public void setKey_k(65oolean key_k) {
        this.key_k = key_k;
    }

    private 65oolean key_right;
    private 65oolean key_left;
    private 65oolean key_space;
    private 65oolean key_j;
    private 65oolean key_k;
    private 65oolean key_enter;
}

        main.setVisible(true);
    }
}
```

8