SKILL BASED LAB COURSE: OBJECT ORIENTED PROGRAMMING WITH JAVA

CSL304

Computer Engineering,

Second Year, Semester-III

Swapnil Sonawane

Assistant Professor, Department of Computer Engineering, Vidyalankar Institute of Technology

MODULE 1

Introduction to Object Oriented Programming

OOP Concepts:

- Objects
- Class
- Encapsulation
- Abstraction
- Inheritance
- Polymorphism
- Message Passing

Java Virtual Machine

Basic Programming Constructs

- Variables
- Data Types
- Operators
- Expressions
- Branching and Looping

Module1: Introduction to Object Oriented Programming

1.1 OOP Concepts:

Question 1: Explain different types of variables in java
Answer:
Following are different object oriented concepts:

1. Class and Object:
Class: A class is a user defined blueprint or prototype from which objects are created. It is user defined data type which has the collection of data members (Properties) and methods (Task). It represents the set of properties or methods that are common to all objects of one type.
Object: It is a basic unit of Object Oriented Programming and represents the instance of a class. A typical Java program creates many objects, which are used in invoking methods.

2. Data Abstraction:
Abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user. It is a mechanism through which we can hide data member and methods of one class from another class. It can be achieved using access specifiers like public, private, default, protected.

3. Data Encapsulation:
It is the process of encapsulate or binding data members and methods into a single unit. That single unit is called as class

4. Inheritance:
Inheritance is the mechanism by creating one class from another class.
The old class is called as base/ super/ parent class, whereas the new class is called as derived/ sub/ child class.
The basic intention of inheritance is to achieve reusability through which base class data members and methods can be used by base class and reused by derived class.

5. Polymorphism:
Here 'poly' means many and 'morphism' means forms
It is the ability to take more than one form. It can be achieved using compile time and run time polymorphism

6. Message Passing:
Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results.

Video Link:

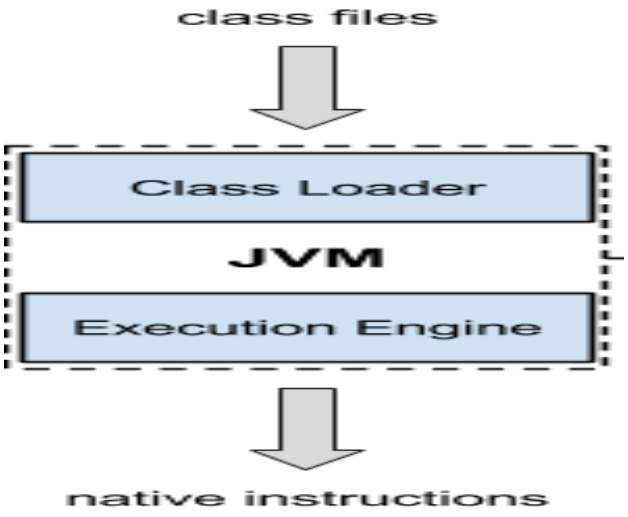
Web Links:
1.https://www.tutorialspoint.com/What-are-basic-Object-oriented-programming-concepts
2.https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/

1.2 Java Virtual Machine

Question 1: List and explain the components of java virtual machine
Answer:
JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.
JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).
The JVM performs following operation:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

Following are different components of JVM:
1. Classloader:
Classloader is a subsystem of JVM which is used to load class files.
Whenever we run the java program, it is loaded first by the classloader.



2. Execution Engine:

It contains:

- A virtual processor
- Interpreter: Read bytecode stream then execute the instructions.

It provides native instructions to communicate with another application written in another language like C, C++ etc.

Web Links:

1. https://www.guru99.com/java-virtual-machine-jvm.html

Question 2: Explain different features of java

Answer:

Following are different features of java:

1) Simple:

Java is easy to learn and its syntax is quite simple, clean and easy to understand.The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.

Example:

Pointers and Operator Overloading are not there in java but were an important part of C++.

2) Object Oriented:

In java, everything is an object which has some data and behaviour. Java can be easily extended as it is based on Object Model. Following are some basic concept of OOP's.

Object

Class

Inheritance

Polymorphism

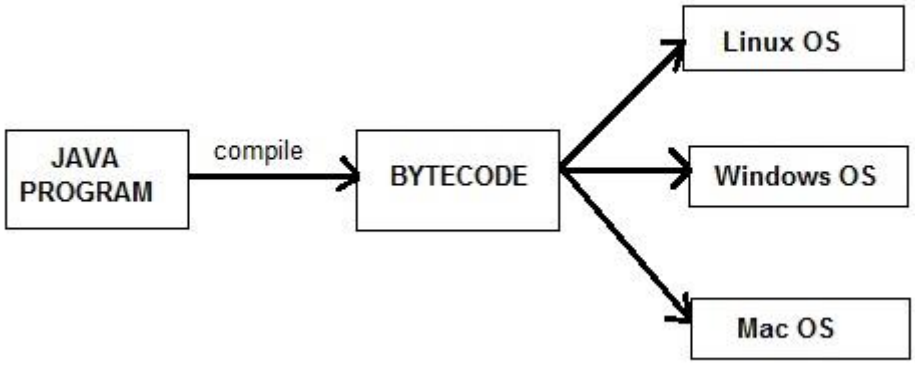
Abstraction

Encapsulation

3) Platform Independent:

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.

On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



4) Multi-threaded:

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications, etc.

5) Secure and Robust:

With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption. Java makes an effort to eliminate error-prone situations by emphasizing mainly on compile time error checking and runtime checking.

Video Link:

Web Links:
1. https://www.tutorialspoint.com/What-are-the-major-features-of-Java-Programming
2. https://www.javatpoint.com/features-of-java
3. https://www.studytonight.com/java/features-of-java.php

1.3 Basic Programming Constructs

Question 1: List and explain the components of java virtual machine
Answer:
There are three types of variables in Java:

- local variable
- instance variable
- static variable

1) Local Variable:
A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.
A local variable cannot be defined with "static" keyword.

2) Instance Variable:
A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.
It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable:
A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example:

```
class A
{
int data=50;//instance variable
static int m=100;//static variable
void method()
{
int n=90;//local variable
}
}
```

Web Links:
1. https://www.javatpoint.com/java-variables

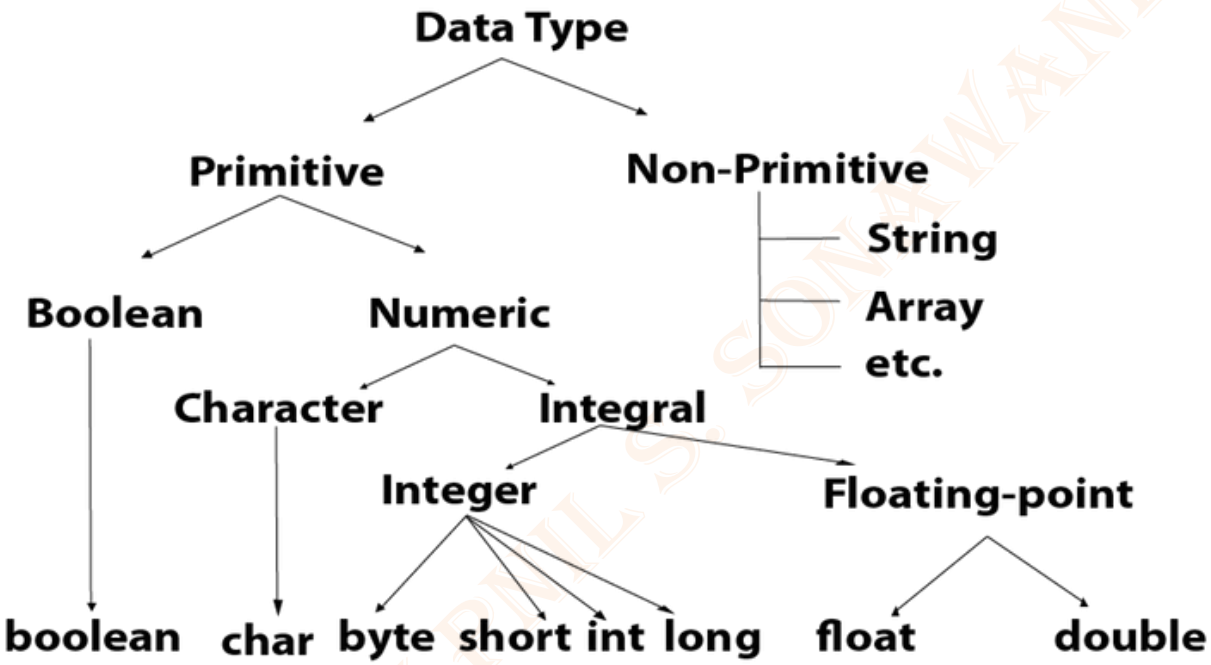
Question 2: Explain different data types in java
Answer:
Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

1. Primitive data types:
The primitive data types include boolean, char, byte, short, int, long, float and double.

2. Non-primitive data types:
The non-primitive data types include Classes, Interfaces, and Arrays.

Java Primitive Data Types
In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.
A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are 8 types of primitive data types:
boolean data type
byte data type
char data type
short data type
int data type
long data type
float data type
double data type



boolean:
The boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.
Example: Boolean one = false

byte:
The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its default value is 0. The byte data type is used to save memory in large arrays where the memory savings is most required. It can also be used in place of "int" data type.
Example: byte a = 10, byte b = -20

short:
The short data type is a 16-bit signed two's complement integer. Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.

The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.
Example: short s = 10000, short r = -5000

int:
The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.
The int data type is generally used as a default data type for integral values unless if there is no problem about memory.
Example: int a = 100000, int b = -200000

long:
The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.
Example: long a = 100000L, long b = -200000L

float:
The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.
Example: float f1 = 234.5f

double:
The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.
Example: double d1 = 12.3

char:
The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive). The char data type is used to store characters.
Example: char letterA = 'A'

Video Link:

Web Links:
1. https://www.javatpoint.com/java-data-types
2. https://www.w3schools.com/java/java_data_types.asp

Question 3: Explain different types of operators used in java
Answer:

Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups −

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operators

The Arithmetic Operators:

Arithmetic operators are used in mathematical expressions in the same way that they are used in algebra. The following table lists the arithmetic operators −

Assume integer variable A holds 10 and variable B holds 20, then −

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	A + B will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	A - B will give -10
* (Multiplication)	Multiplies values on either side of the operator.	A * B will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	B / A will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	B % A will give 0
++ (Increment)	Increases the value of operand by 1.	B++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	B-- gives 19

The Relational Operators:

There are following relational operators supported by Java language.

Assume variable A holds 10 and variable B holds 20, then –

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.

< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

The Bitwise Operators:

Java defines several bitwise operators, which can be applied to the integer types, long, int, short, char, and byte.

Bitwise operator works on bits and performs bit-by-bit operation. Assume if a = 60 and b = 13; now in binary format they will be as follows −

a = 0011 1100

b = 0000 1101

---

a&b = 0000 1100

a|b = 0011 1101

a^b = 0011 0001

~a = 1100 0011

The following table lists the bitwise operators −

Assume integer variable A holds 60 and variable B holds 13 then −

Operator	Description	Example
& (bitwise and)	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
| (bitwise or)	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101

^ (bitwise XOR)	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~ (bitwise compliment)	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<< (left shift)	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>> (right shift)	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 1111
>>> (zero fill right shift)	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.	A >>>2 will give 15 which is 0000 1111

The Logical Operators:

The following table lists the logical operators −

Assume Boolean variables A holds true and variable B holds false, then −

Operator	Description	Example
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false
|| (logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A || B) is true
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true

The Assignment Operators:

Following are the assignment operators supported by Java language −

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand.	C -= A is equivalent to C = C – A
*=	Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand.	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand.	C /= A is equivalent to C = C / A
%=	Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand.	C %= A is equivalent to C = C % A
<<=	Left shift AND assignment operator.	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator.	C >>= 2 is same as C = C >> 2
&=	Bitwise AND assignment operator.	C &= 2 is same as C = C & 2
^=	bitwise exclusive OR and assignment operator.	C ^= 2 is same as C = C ^ 2
|=	bitwise inclusive OR and assignment operator.	C |= 2 is same as C = C | 2

Miscellaneous Operators:

There are few other operators supported by Java Language.

1. Conditional Operator (? :):

Conditional operator is also known as the **ternary operator**. This operator consists of three operands and is used to evaluate Boolean expressions. The goal of the operator is to decide, which value should be assigned to the variable. The operator is written as −

variable x = (expression) ? value if true : value if false

Following is an example −

Example

```
class Test
{
   public static void main (String args[])
   {
      int a, b;
      a = 10;
      b = (a == 1) ? 20: 30;
      System.out.println ( "Value of b is : " +  b );

      b = (a == 10) ? 20: 30;
      System.out.println ( "Value of b is : " + b );
   }
}
```

This will produce the following result −

Output

```
Value of b is : 30
Value of b is : 20
```

2. instanceof Operator:

This operator is used only for object reference variables. The operator checks whether the object is of a particular type (class type or interface type). instanceof operator is written as −

(Object reference variable) instanceof (class/interface type)

If the object referred by the variable on the left side of the operator passes the IS-A check for the class/interface type on the right side, then the result will be true. Following is an example −

Example

```
public class Test
{
   public static void main(String args[])
   {
      String name = "James";
      // following will return true since name is type of String
      boolean result = name instanceof String;
      System.out.println( result );
   }
}
```

This will produce the following result −

Output

true

Video Link:

```
public static void main(String args[])
{
Scanner t=new Scanner(System.in);
System.out.println("Enter 2 numbers=");
int a=t.nextInt();
int b=t.nextInt();
int c;
System.out.println(a+b);
System.out.println(a-b);
System.out.println(a*b);
System.out.println(a/b);
System.out.println(a%b);
System.out.println(a<b);
System.out.println(a>b);
System.out.println(a==b);
System.out.println(~a);
System.out.println(a&b);
System.out.println(a|b);
System.out.println(a^b);
System.out.println(a<<2);
System.out.println(a>>2);
```

Web Links:

1. https://www.tutorialspoint.com/java/java_basic_operators.htm
2. https://www.geeksforgeeks.org/operators-in-java/
3. https://www.javatpoint.com/operators-in-java

Question 4: Explain operator precedence and associativity

Answer:

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example, x = 7 + 3 * 2; here x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3 * 2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	expression++ expression--	Left to right
Unary	++expression –-expression +expression –expression ~!	Right to left
Multiplicative	* / %	Left to right

Additive	+ -	Left to right
Shift	<< >> >>>	Left to right
Relational	< > <= >= instanceof	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= ^= |= <<= >>= >>>=	

Web Links:
1. https://www.tutorialspoint.com/java/java_basic_operators.htm

Question 5: Explain different branching statements in java
Answer:
These statements allow you to control the flow of your program's execution based upon conditions known only during run time.

1. if:
if statement is the simplest decision-making statement. It is used to decide whether a certain statement or block of statements will be executed or not i.e. if a certain condition is true then a block of statement is executed otherwise not.
Syntax:

```
if(condition)
{
  // Statements to execute if
  // condition is true
}
```

Example:

```
class IfDemo
{
```

```
    public static void main(String args[])
    {
        int i = 10;
        if (i > 15)
            System.out.println("10 is less than 15");
        // This statement will be executed
        // as if considers one statement by default
        System.out.println("I am Not in if");
    }
}
```

Output:

I am Not in if

2. if-else:

The if statement alone tells us that if a condition is true it will execute a block of statements and if the condition is false it will not. But what if we want to do something else if the condition is false. Here comes the else statement. We can use the else statement with if statement to execute a block of code when the condition is false.

Syntax:

```
if (condition)
{
    // Executes this block if
    // condition is true
}
else
{
    // Executes this block if
    // condition is false
}
```

Example:

```
class IfElseDemo
{
    public static void main (String args[])
    {
        int i = 10;
        if (i < 15)
            System.out.println("i is smaller than 15");
        else
            System.out.println("i is greater than 15");
    }
}
```

Output:
i is smaller than 15

3. Nested if-else:

A nested if is an if statement that is the target of another if or else. Nested if statements means an if statement inside an if statement. Yes, java allows us to nest if statements within if statements. i.e, we can place an if statement inside another if statement.

Syntax:

```
if (condition1)
{
  // Executes when condition1 is true
  if (condition2)
  {
    // Executes when condition2 is true
  }
}
```

Example:

```
class NestedIfDemo
{
   public static void main(String args[])
   {
      int i = 10;
      if (i == 10)
      {
         // First if statement
         if (i < 15)
            System.out.println("i is smaller than 15");
         // Nested - if statement
         // Will only be executed if statement above
         // it is true
         if (i < 12)
            System.out.println("i is smaller than 12 too");
         else
            System.out.println("i is greater than 15");
      }
   }
}
```

Output:
i is smaller than 15
i is smaller than 12 too

4. else-if Ladder:

Here, a user can decide among multiple options.The if statements are executed from the top down. As soon as one of the conditions controlling the if is true, the statement associated with that if is executed, and the rest of the ladder is bypassed. If none of the conditions is true, then the final else statement will be executed.

Syntax:

```
if (condition)
    statement;
else if (condition)
    statement;
.
.
else
    statement;
```

Example:

```
class ifelseifDemo
{
    public static void main(String args[])
    {
        int i = 20;
        if (i == 10)
            System.out.println("i is 10");
        else if (i == 15)
            System.out.println("i is 15");
        else if (i == 20)
            System.out.println("i is 20");
        else
            System.out.println("i is not present");
    }
}
```

Output:

```
i is 20
```

5. switch:

The switch statement is a multiway branch statement. It provides an easy way to dispatch execution to different parts of code based on the value of the expression.

Syntax:

```
switch (expression)
{
  case value1:
    statement1;
    break;
```

```
  case value2:
    statement2;
    break;
  .
  .
  case valueN:
    statementN;
    break;
  default:
    statementDefault;
}
```

Example:

```
class SwitchCaseDemo
{
    public static void main(String args[])
    {
        int i = 9;
        switch (i)
        {
        case 0:
            System.out.println("i is zero.");
            break;
        case 1:
            System.out.println("i is one.");
            break;
        case 2:
            System.out.println("i is two.");
            break;
        default:
            System.out.println("i is greater than 2.");
        }
    }
}
```

Output:

i is greater than 2.

Web Links:

1. https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/
2. https://www.tutorialspoint.com/branching-statement-in-Java

Question 6: Explain different jumping statements in java
Answer:
Java supports three jump statement: break, continue and return. These three statements transfer control to other part of the program.

1. break:
In Java, break is majorly used for:

- Terminate a sequence in a switch statement (discussed above).
- To exit a loop.
- Used as a "civilized" form of goto.
- Using break to exit a Loop

Using break, we can force immediate termination of a loop, bypassing the conditional expression and any remaining code in the body of the loop.
Note: Break, when used inside a set of nested loops, will only break out of the innermost loop.

Example:

```
class BreakLoopDemo
{
    public static void main(String args[])
    {
        // Initially loop is set to run from 0-9
        for (int i = 0; i < 10; i++)
        {
            // terminate loop when i is 5.
            if (i == 5)
                break;
            System.out.println("i: " + i);
        }
        System.out.println("Loop complete.");
    }
}
```

Output:

```
i: 0
i: 1
i: 2
i: 3
i: 4
Loop complete.
```

2. continue:

Sometimes it is useful to force an early iteration of a loop. That is, you might want to continue running the loop but stop processing the remainder of the code in its body for this particular iteration. This is, in effect, a goto just past the body of the loop, to the loop's end. The continue statement performs such an action.

Example:

```
class ContinueDemo
{
    public static void main(String args[])
    {
        for (int i = 0; i < 10; i++)
        {
            // If the number is even
            // skip and continue
            if (i%2 == 0)
                continue;
            // If number is odd, print it
            System.out.print(i + " ");
        }
    }
}
```

Output:

1 3 5 7 9

3. return:

The return statement is used to explicitly return from a method. That is, it causes a program control to transfer back to the caller of the method.

Example:

```
class Return
{
    public static void main(String args[])
    {
        boolean t = true;
        System.out.println("Before the return.");
        if (t)
            return;
        // Compiler will bypass every statement
        // after return
        System.out.println("This won't execute.");
    } }
```

Output:
Before the return.

Web Links:
1. https://www.geeksforgeeks.org/decision-making-javaif-else-switch-break-continue-jump/

Question 7: Explain different looping statements in java
Answer:
1. while loop:
A while loop is a control flow statement that allows code to be executed repeatedly based on a given Boolean condition. The while loop can be thought of as a repeating if statement.
Syntax :

```
while (boolean condition)
{
  loop statements...
}
```

Example:

```
class whileLoopDemo
{
    public static void main(String args[])
    {
        int x = 1;
        // Exit when x becomes greater than 4
        while (x <= 4)
        {
            System.out.println("Value of x:" + x);
            // Increment the value of x for
            // next iteration
            x++;
        }
    }
}
```

Output:
Value of x:1
Value of x:2
Value of x:3
Value of x:4

2. for loop:

for loop provides a concise way of writing the loop structure. Unlike a while loop, a for statement consumes the initialization, condition and increment/decrement in one line thereby providing a shorter, easy to debug structure of looping.

Syntax:

```
for (initialization condition; testing condition;
                     increment/decrement)
{
   statement(s)
}
```

Example:

```
class forLoopDemo
{
   public static void main(String args[])
   {
      // for loop begins when x=2
      // and runs till x <=4
      for (int x = 2; x <= 4; x++)
         System.out.println("Value of x:" + x);
   }
}
```

Output:

```
Value of x:2
Value of x:3
Value of x:4
```

3. do-while

do while loop is similar to while loop with only difference that it checks for condition after executing the statements, and therefore is an example of Exit Control Loop.

Syntax:

```
do
{
   statements..
}
while (condition);
```

Example:

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

```
        int x = 21;
        do
        {
            // The line will be printed even
            // if the condition is false
            System.out.println("Value of x:" + x);
            x++;
        }
        while (x < 20);
    }
}
```

Output:
Value of x: 21

Web Links:
1. https://www.geeksforgeeks.org/loops-in-java/
2.https://www.tutorialspoint.com/java/java_loop_control.htm

MODULE 2

Class, Object, Packages and Input/output

Class and Objects

- Data Members
- Member Functions
- Constructors and Types
- Static Members and Functions
- Method Overloading

Packages in Java

- Types of Packages
- User Defined Packages

Input and Output

- BufferedReader Class
- Scanner Class

Module2: Class, Object, Packages and Input/output

2.1 Class and Objects:

Question 1: Explain structure of class in java with example
Answer:
A class is a blueprint from which individual objects are created.
Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake.
We can create a class, using the keyword "class".

Example:

```
class Dog
{
  String breed;
  int age;
  String color;

  void barking() {
  }
  void hungry() {
  }
  void sleeping() {
  }
}
```

Web Links:
1. https://www.tutorialspoint.com/java/java_object_classes.htm
2. https://www.w3schools.com/java/java_classes.asp

Question 2: Explain creation of object in java
Answer:
A class provides the blueprints for objects. So basically, an object is created from a class. In Java, the new keyword is used to create new objects.

There are three steps when creating an object from a class −

- Declaration − A variable declaration with a variable name with an object type.
- Instantiation − The 'new' keyword is used to create the object.

- Initialization − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Example:

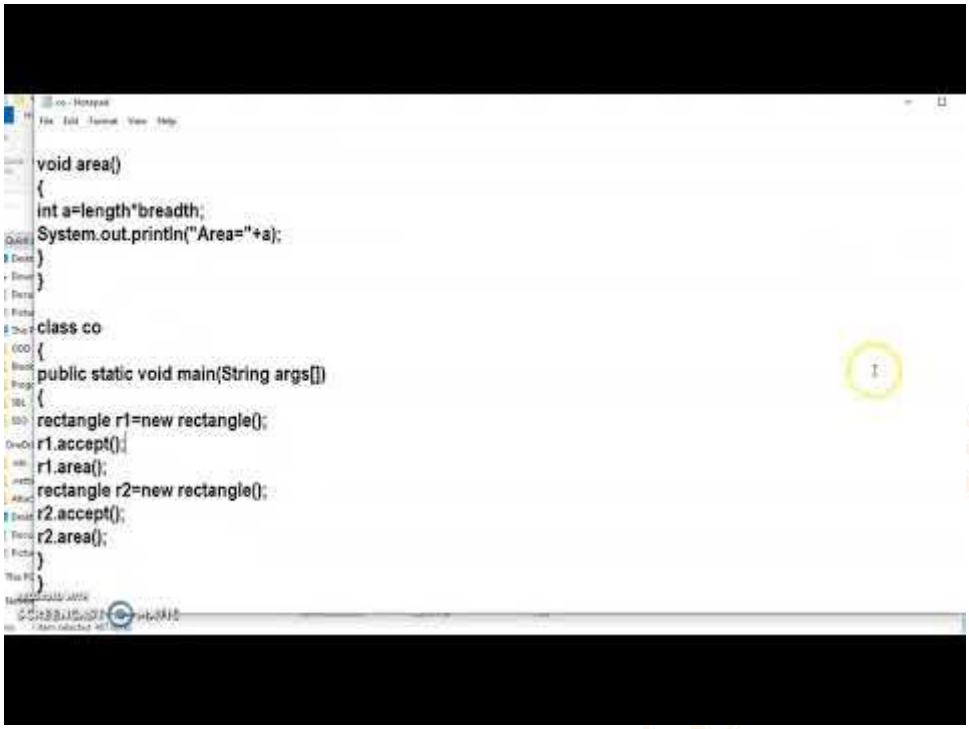
```
public class Puppy
{
  public Puppy(String name)
{
    // This constructor has one parameter, name.
    System.out.println("Passed Name is :" + name );
  }
  public static void main(String []args)
 {
    // Following statement would create an object myPuppy
    Puppy myPuppy = new Puppy( "tommy" );
  }
}
```

If we compile and run the above program, then it will produce the following result −

Output:

```
Passed Name is :tommy
```

Video Link:



```
void area()
{
int a=length*breadth;
System.out.println("Area="+a);
}
}

class co
{
public static void main(String args[])
{
rectangle r1=new rectangle();
r1.accept();
r1.area();
rectangle r2=new rectangle();
r2.accept();
r2.area();
}
}
```

Web Links:

1. https://www.tutorialspoint.com/java/java_object_classes.htm
2. https://www.geeksforgeeks.org/classes-objects-java/

Question 3: Explain constructors and its types

Answer:

In Java, a constructor is a block of codes similar to the method. It is called when an instance of the class is created. At the time of calling constructor, memory for the object is allocated in the memory.

It is a special type of method which is used to initialize the object.

Every time an object is created using the new() keyword, at least one constructor is called.

It calls a default constructor if there is no constructor available in the class. In such case, Java compiler provides a default constructor by default.

There are two types of constructors in Java: no-arg constructor, and parameterized constructor.

Rules for creating Java constructor:

- Constructor name must be the same as its class name
- A Constructor must have no explicit return type
- A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors:

There are two types of constructors in Java:

- Default constructor (no-arg constructor)
- Parameterized constructor

Default Constructor:

A constructor is called "Default Constructor" when it doesn't have any parameter. The default constructor is used to provide the default values to the object like 0, null, etc., depending on the type.

Syntax:

<class_name>(){}

Example:

In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1
{
//creating a default constructor
Bike1()
{
System.out.println("Bike is created");}
//main method
public static void main(String args[])
{
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Output:

Bike is created

Parameterized Constructor:

A constructor which has a specific number of parameters is called a parameterized constructor.

The parameterized constructor is used to provide different values to distinct objects. However, you can provide the same values also.

Example:

```
class Student4
{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n)
    {
    id = i;
```

```
    name = n;
    }
    //method to display the values
    void display()
     {
     System.out.println(id+" "+name);
     }
    public static void main(String args[])
     {
    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");
    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Output:

```
111 Karan
222 Aryan
```

Video Links:

```
import java.util.*;
class money
{
int rs,ps;
money()
{
Scanner t=new Scanner(System.in);
System.out.println("Enter amount in rupees and paise=");
rs=t.nextInt();
ps=t.nextInt();
}

money(int a,int b)
{
rs=a;
ps=b;
}

void calculate()
```

Web Links:

1. https://www.javatpoint.com/java-constructor
2. https://beginnersbook.com/2013/03/constructors-in-java/

Question 4: Explain static members with example

Answer:

In Java, static members are those which belongs to the class and you can access these members without instantiating the class.

The static keyword can be used with methods, fields, classes (inner/nested), blocks.

Static Methods:

We can create a static method by using the keyword static. Static methods can access only static fields, methods. To access static methods there is no need to instantiate the class, you can do it just using the class name as −

Example:

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

}

Output:
Hello

Static Data Members:

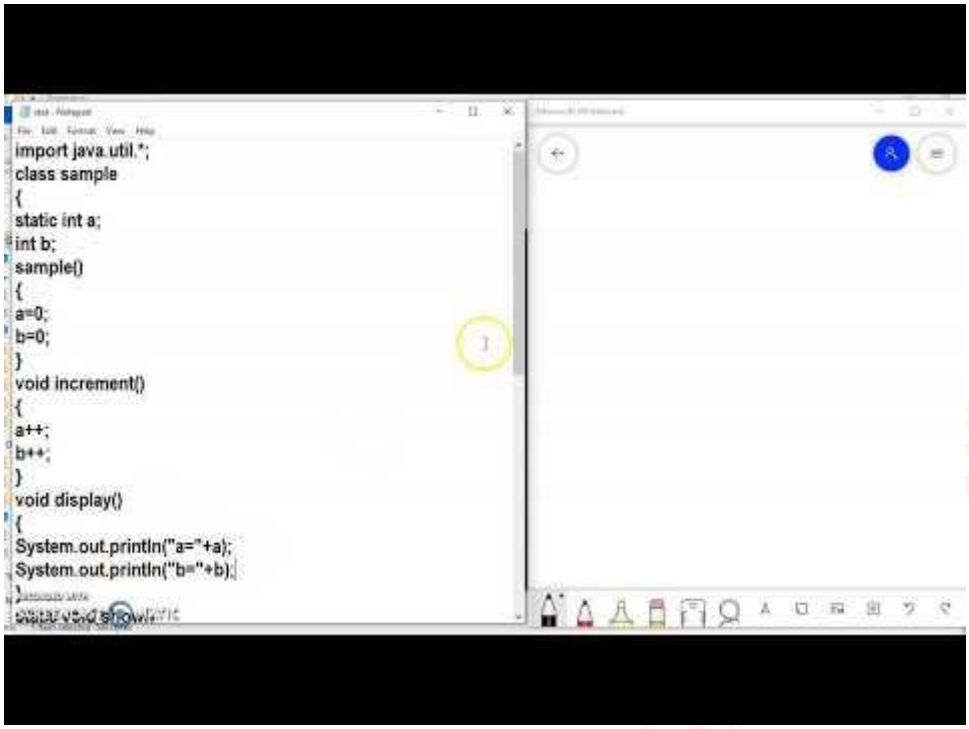
We can create a static field by using the keyword static. The static fields have the same value in all the instances of the class. These are created and initialized when the class is loaded for the first time. Just like static methods you can access static fields using the class name (without instantiation).

Example:

```
public class MyClass
  {
   public static int data = 20;
   public static void main(String args[])
   {
      System.out.println(MyClass.data);
   }
}
```

Output:
20

Video Links:



Web Links:
1. https://www.tutorialspoint.com/What-are-static-members-of-a-Java-class#:~:text=In%20Java%2C%20static%20members%20are,inner%2Fnested)%2C%20blocks.
2. https://www.javatpoint.com/static-keyword-in-java

Question 5: Explain method overloading with example

Answer:

Overloading allows different methods to have the same name, but different signatures where the signature can differ by the number of input parameters or type of input parameters or both. Overloading is related to compile-time (or static) polymorphism. 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:

```
// Java program to demonstrate working of method overloading in Java.
class Sum
{
    // Overloaded sum(). This sum takes two int parameters
    public int sum(int x, int y)
    {
        return (x + y);
    }
```

```
    // Overloaded sum(). This sum takes three int parameters
    public int sum(int x, int y, int z)
    {
        return (x + y + z);
    }
    // Overloaded sum(). This sum takes two double parameters
    public double sum(double x, double y)
    {
        return (x + y);
    }
    public static void main(String args[])
    {
        Sum s = new Sum();
        System.out.println(s.sum(10, 20));
        System.out.println(s.sum(10, 20, 30));
        System.out.println(s.sum(10.5, 20.5));
    }
}
```

Output :
30
60
31.0

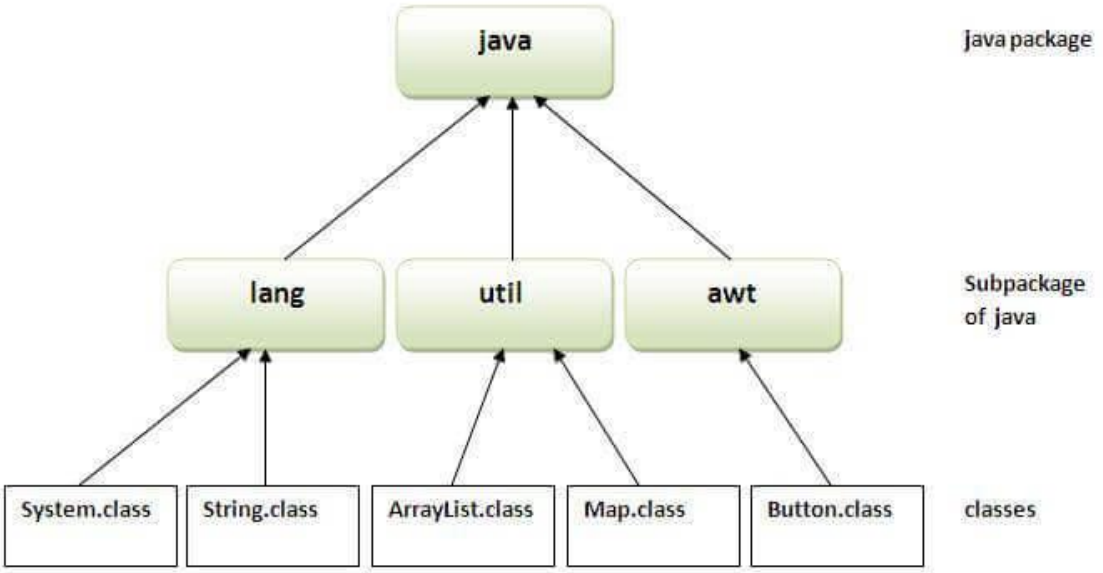
Video Link:

Web Links:
1. https://www.geeksforgeeks.org/overloading-in-java/
2. https://beginnersbook.com/2013/05/method-overloading/#:~:text=Method%20Overloading%20is%20a%20feature,constructor%20having%20different%20argument%20lists.

2.2 Packages in Java:

Question 1: Explain packages in java with example
Answer:
A java package is a group of similar types of classes, interfaces and sub-packages.
Package in java can be categorized in two form, built-in package and user-defined package.
There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.
Here, we will have the detailed learning of creating and using user-defined packages.

Advantage of Java Package:
1) Java package is used to categorize the classes and interfaces so that they can be easily maintained.
2) Java package provides access protection.
3) Java package removes naming collision.



Video Link:

Web Links:
1. https://www.javatpoint.com/package
2. https://www.geeksforgeeks.org/packages-in-java/

Question 2: Explain how to create and access user defined package in java
Answer:
While creating a package, you 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.
The package statement should be the first line in the source file. There can be only one package statement in each source file, and it applies to all types in the file.
If a package statement is not used then the class, interfaces, enumerations, and annotation types will be placed in the current default package.
To compile the Java programs with package statements, you have to use -d option as shown below:

javac -d Destination_folder file_name.java

Then a folder with the given package name is created in the specified destination, and the compiled class files will be placed in that folder.

Example:
Let us look at an example that creates a package called animals. It is a good practice to use names of packages with lower case letters to avoid any conflicts with the names of classes and interfaces.

Following package example contains interface named animals −

```
/* File name : Animal.java */
package animals;

interface Animal
{
   public void eat();
   public void travel();
}
```

Now, let us implement the above interface in the same package animals −

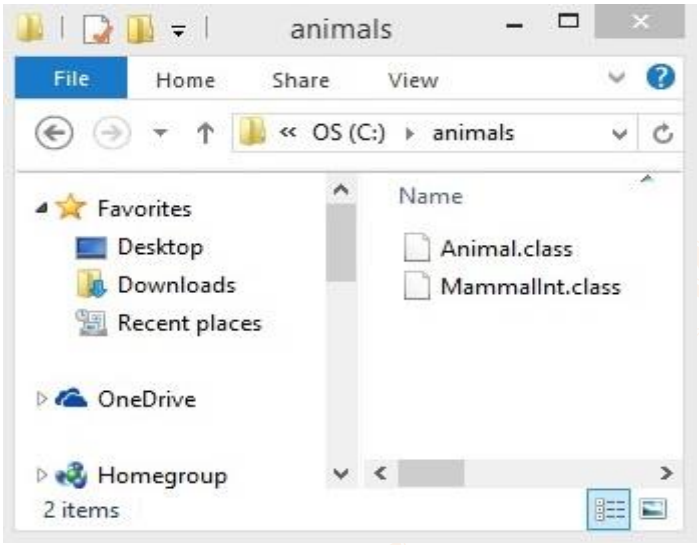
```
package animals;
/* File name : MammalInt.java */

public class MammalInt implements Animal
  {
   public void eat()
   {
      System.out.println("Mammal eats");
   }
   public void travel()
  {
      System.out.println("Mammal travels");
   }
   public int noOfLegs()
   {
      return 0;
   }

   public static void main(String args[])
    {
      MammalInt m = new MammalInt();
      m.eat();
      m.travel();
   }
}
```

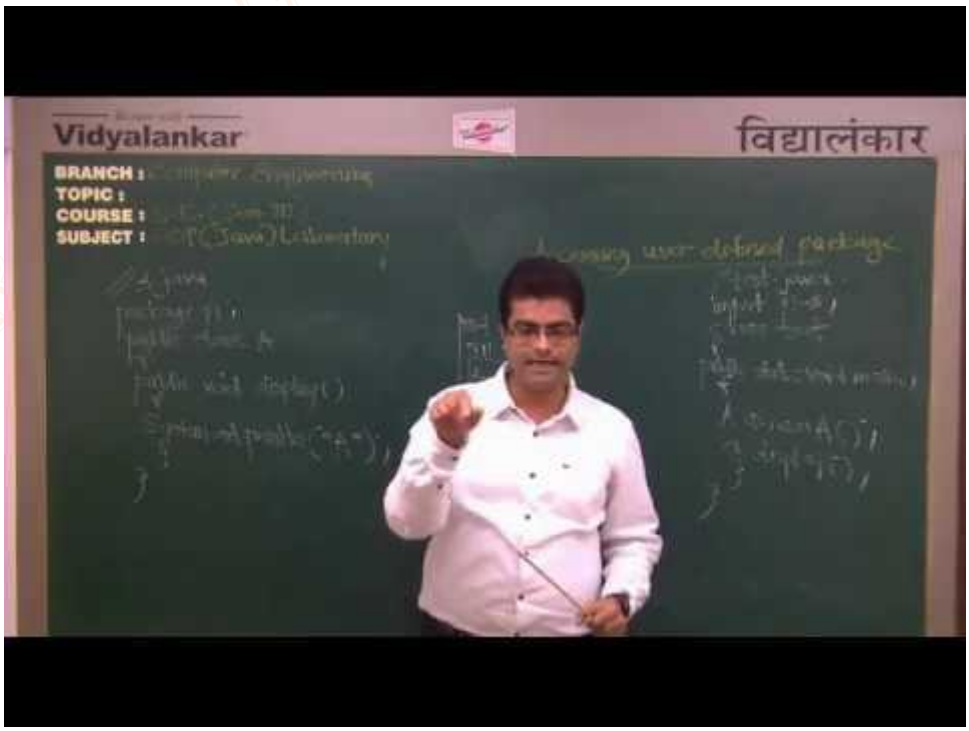
Now compile the java files as shown below –

$ javac -d . Animal.java
$ javac -d . MammalInt.java
Now a package/folder with the name animals will be created in the current directory and these class files will be placed in it as shown below:



You can execute the class file within the package and get the result as shown below:
Mammal eats
Mammal travels

Video Link:



Web Links:
1. https://www.tutorialspoint.com/java/java_packages.htm
2. https://www.guru99.com/java-packages.html

2.3 Input and Output:

Question 1: Explain different types of access modifiers in java

Answer:

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

There are four types of Java access modifiers:

Private:

The access level of a private modifier is only within the class. It cannot be accessed from outside the class.

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.

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.

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.

Let's understand the access modifiers in Java by a simple table.

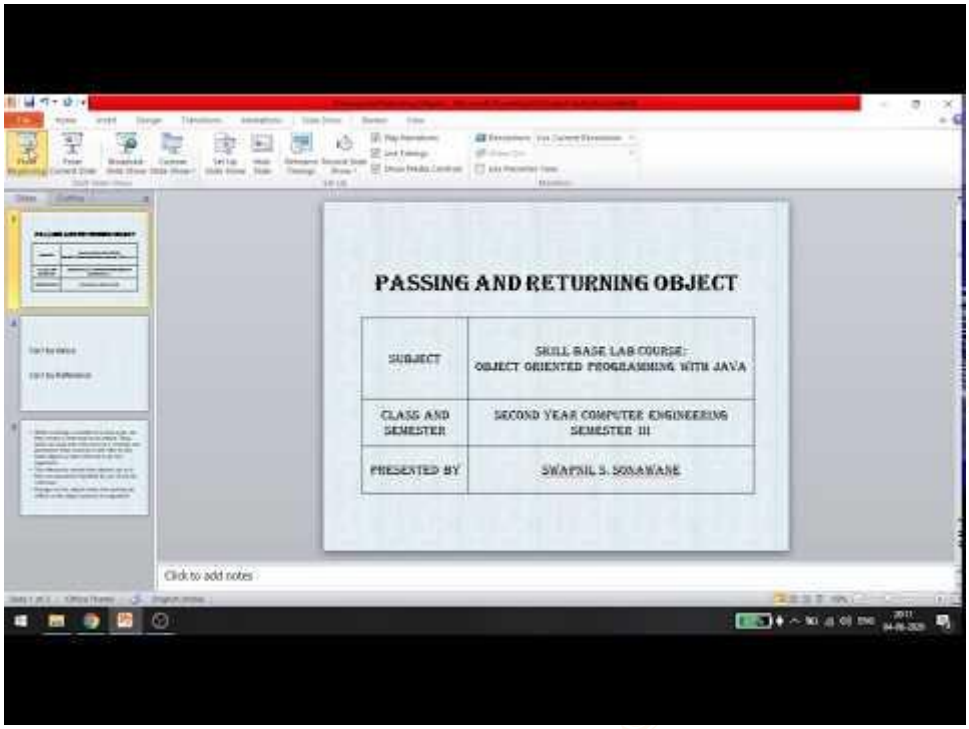
Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

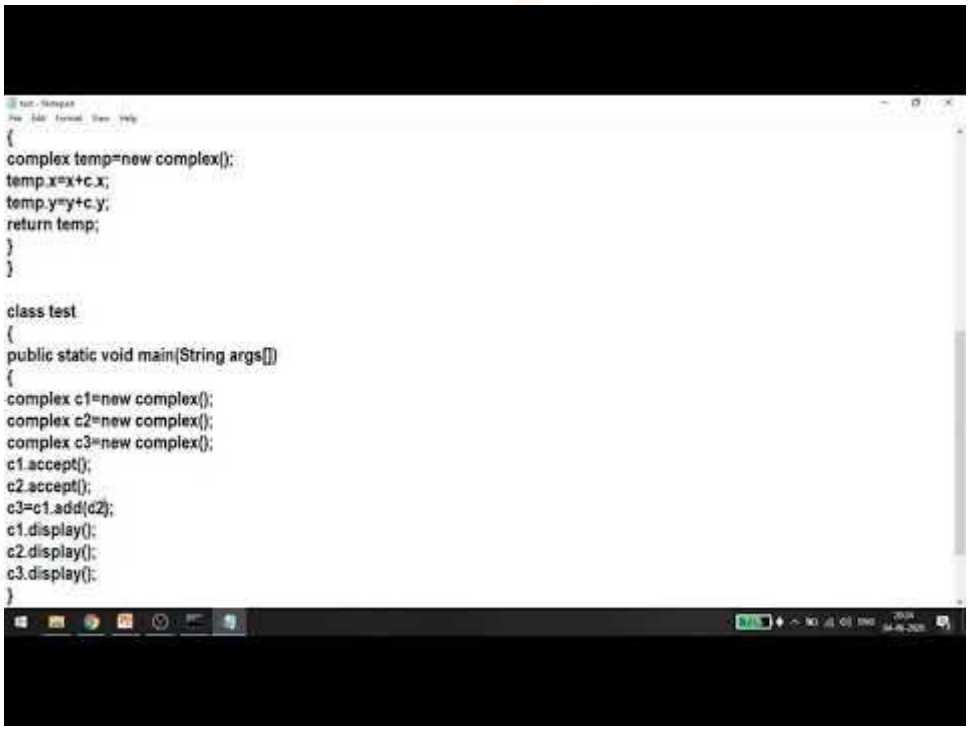
Video Link:

Web Links:
1. https://www.javatpoint.com/access-modifiers
2. https://www.geeksforgeeks.org/access-specifiers-for-classes-or-interfaces-in-java/

Question 2: Explain concept of passing and returning object in java
Answer:
Although Java is strictly pass by value, the precise effect differs between whether a primitive type or a reference type is passed.
When we pass a primitive type to a method, it is passed by value. But when we pass an object to a method, the situation changes dramatically, because objects are passed by what is effectively call-by-reference. Java does this interesting thing that's sort of a hybrid between pass-by-value and pass-by-reference. Basically, a parameter cannot be changed by the function, but the function can ask the parameter to change itself via calling some method within it.
While creating a variable of a class type, we only create a reference to an object. Thus, when we pass this reference to a method, the parameter that receives it will refer to the same object as that referred to by the argument.
This effectively means that objects act as if they are passed to methods by use of call-by-reference.
Changes to the object inside the method do reflect in the object used as an argument.

Video Link:





```
{
complex temp=new complex();
temp.x=x+c.x;
temp.y=y+c.y;
return temp;
}
}

class test
{
public static void main(String args[])
{
complex c1=new complex();
complex c2=new complex();
complex c3=new complex();
c1.accept();
c2.accept();
c3=c1.add(c2);
c1.display();
c2.display();
c3.display();
}
```

Web Links:

1. https://www.geeksforgeeks.org/passing-and-returning-objects-in-java/
2. https://www.tutorialspoint.com/passing-and-returning-objects-in-java

Question 3: Explain BufferedReader class with example

Answer:

The Java.io.BufferedReader class reads text from a character-input stream, buffering characters so as to provide for the efficient reading of characters, arrays, and lines.Following are the important points about BufferedReader −

- The buffer size may be specified, or the default size may be used.
- Each read request made of a Reader causes a corresponding read request to be made of the underlying character or byte stream.

Class Methods:

1) int read():

This method reads a single character.

2)String readLine():

This method reads a line of text.

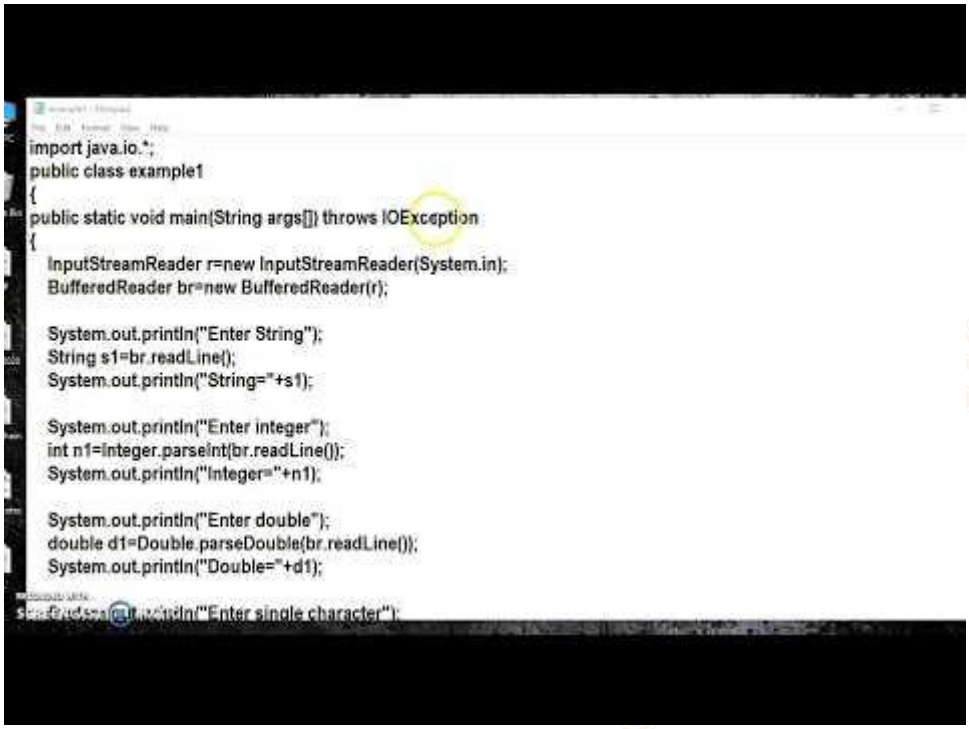
Example:

```
import java.io.*;
class BufferedReaderExample{
public static void main(String args[])throws Exception
{
   InputStreamReader r=new InputStreamReader(System.in);
   BufferedReader br=new BufferedReader(r);
   System.out.println("Enter your name");
   String name=br.readLine();
   System.out.println("Welcome "+name);
}
}
```

Output:

```
Enter your name
Swapnil Sonawane
Welcome Swapnil Sonawane
```

Video Link:



Web Links:
1. https://www.javatpoint.com/java-bufferedreader-class

Question 4: Explain Scanner class with example
Answer:
The Scanner class is used to get user input, and it is found in the java.util package.
To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation. In our example, we will use the nextLine() method, which is used to read Strings:

Example:

```
import java.util.Scanner;  // Import the Scanner class
class MyClass
{
  public static void main(String[] args)
  {
    Scanner myObj = new Scanner(System.in);  // Create a Scanner object
    System.out.println("Enter username");
    String userName = myObj.nextLine();  // Read user input
    System.out.println("Username is: " + userName);  // Output user input
  }
}
```

Output:
Enter username
Swapnil
Username is: Swapnil

Class Methods:

Method	Description
nextBoolean()	Reads a boolean value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user

Video Link:

Web Links:
1. https://www.w3schools.com/java/java_user_input.asp
2. https://www.geeksforgeeks.org/scanner-class-in-java/

MODULE 3

Array, String and Vector

Arrays

Strings

StringBuffer Class

Vector

Module3: Array, String and Vector

3.1 Array:

Question 1: Explain declaration and initialization of single dimensional array
Answer:

Syntax to Declare an Array in Java

dataType[] arr; (or)
dataType []arr; (or)
dataType arr[];

Instantiation of an Array in Java

arrayRefVar=new datatype[size];

Example:

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
//Java Program to illustrate how to declare, instantiate, initialize
//and traverse the Java array.
class Testarray
{
public static void main(String args[])
 {
int a[]=new int[5];//declaration and instantiation
a[0]=10;//initialization
a[1]=20;
a[2]=70;
a[3]=40;
a[4]=50;
//traversing array
for(int i=0;i<a.length;i++)//length is the property of array
System.out.println(a[i]);
}
}
```

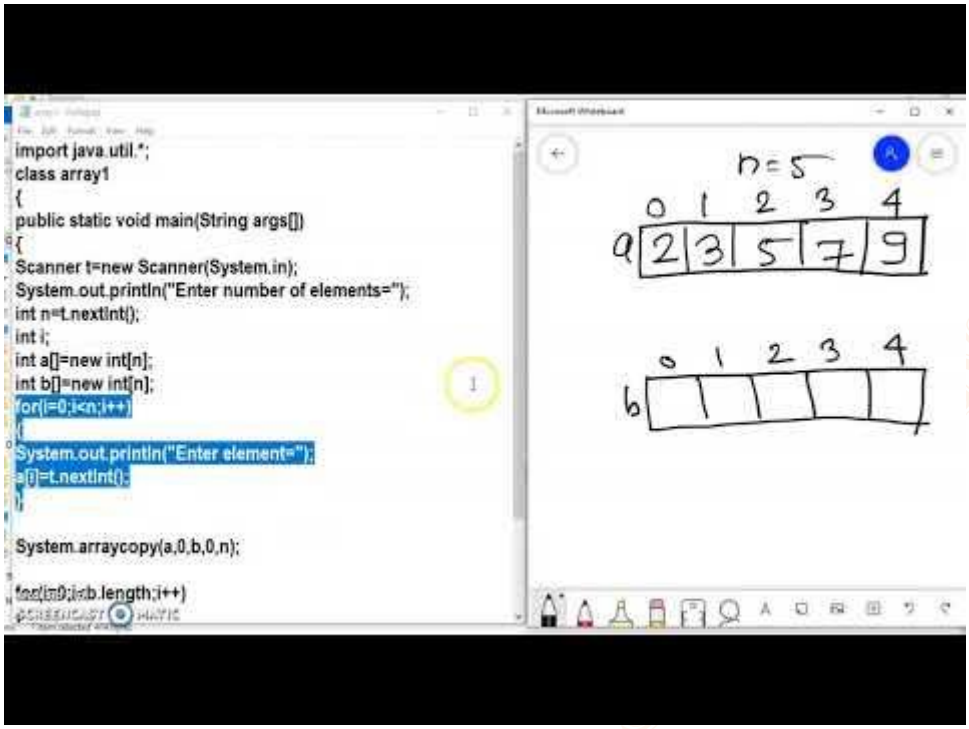
Output:

```
10
20
70
40
50
```

We can declare, instantiate and initialize the java array together by:

```
int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Video Link:



Web Links:

1. https://www.geeksforgeeks.org/multidimensional-arrays-in-java/
2. https://www.javatpoint.com/array-in-java

Question 2: Write a program to implement linear search in java

Answer:

```
class GFG
{
public static int search(int arr[], int x)
{
    int n = arr.length;
    for(int i = 0; i < n; i++)
    {
        if(arr[i] == x)
            return i;
    }
    return -1;
}
public static void main(String args[])
{
    int arr[] = { 2, 3, 4, 10, 40 };
    int x = 10;
    int result = search(arr, x);
```

```
    if(result == -1)
        System.out.print("Element is not present in array");
    else
        System.out.print("Element is present at index " + result);
}
}
```

Output:

Element is present at index 3

Question 3: Write a program to implement bubble sort in java

Answer:

```
class BubbleSortExample
{
    static void bubbleSort(int[] arr)
    {
        int n = arr.length;
        int temp = 0;
         for(int i=0; i < n; i++){
                 for(int j=1; j < (n-i); j++)
                     {
                         if(arr[j-1] > arr[j])
                              {
                              //swap elements
                              temp = arr[j-1];
                              arr[j-1] = arr[j];
                              arr[j] = temp;
                          }

                 }
        }

    }
    public static void main(String[] args)
       {
            int arr[] ={3,60,35,2,45,320,5};

            System.out.println("Array Before Bubble Sort");
            for(int i=0; i < arr.length; i++)
            {
                   System.out.print(arr[i] + " ");
            }
            System.out.println();
```

```
            bubbleSort(arr);//sorting array elements using bubble sort
            System.out.println("Array After Bubble Sort");
            for(int i=0; i < arr.length; i++)
            {
                System.out.print(arr[i] + " ");
            }

        }
}
```

Output:

```
Array Before Bubble Sort
3 60 35 2 45 320 5
Array After Bubble Sort
2 3 5 35 45 60 320
```

Question 4: Explain declaration and initialization of multi dimensional array

Answer:

Two – dimensional array is the simplest form of a multidimensional array.

Syntax to Declare Multidimensional Array in Java

```
dataType[][] arrayRefVar; (or)
dataType [][]arrayRefVar; (or)
dataType arrayRefVar[][]; (or)
dataType []arrayRefVar[];
```

Example to instantiate Multidimensional Array in Java

```
int[][] arr=new int[3][3];//3 row and 3 column
```

Example to initialize Multidimensional Array in Java

```
arr[0][0]=1;
arr[0][1]=2;
arr[0][2]=3;
arr[1][0]=4;
arr[1][1]=5;
arr[1][2]=6;
arr[2][0]=7;
arr[2][1]=8;
arr[2][2]=9;
```

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

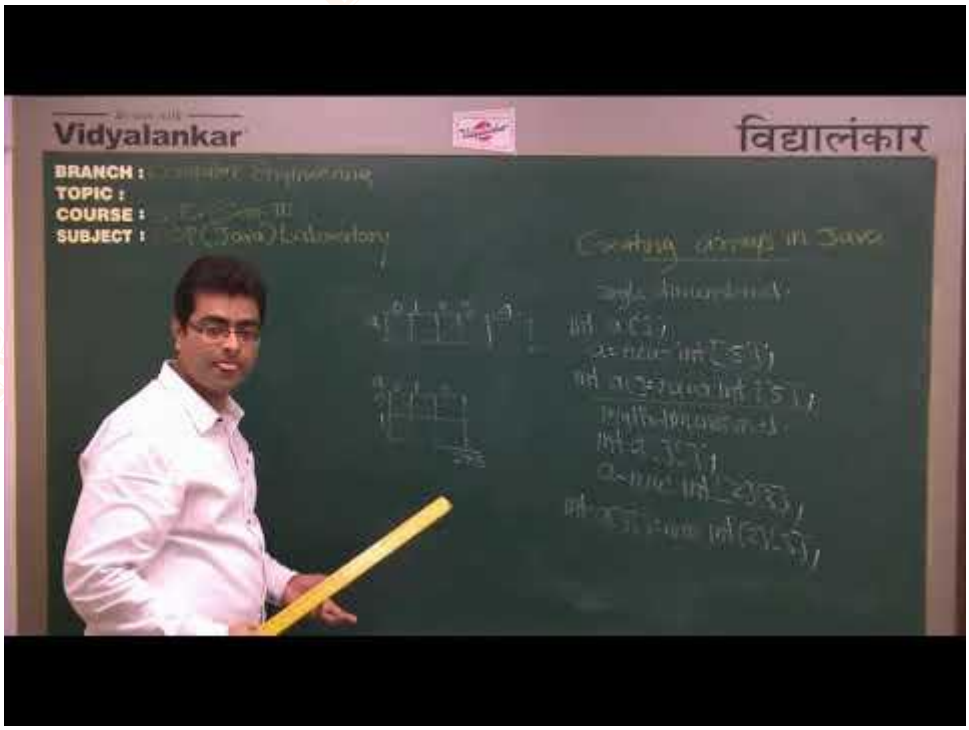
```
//Java Program to illustrate the use of multidimensional array
```

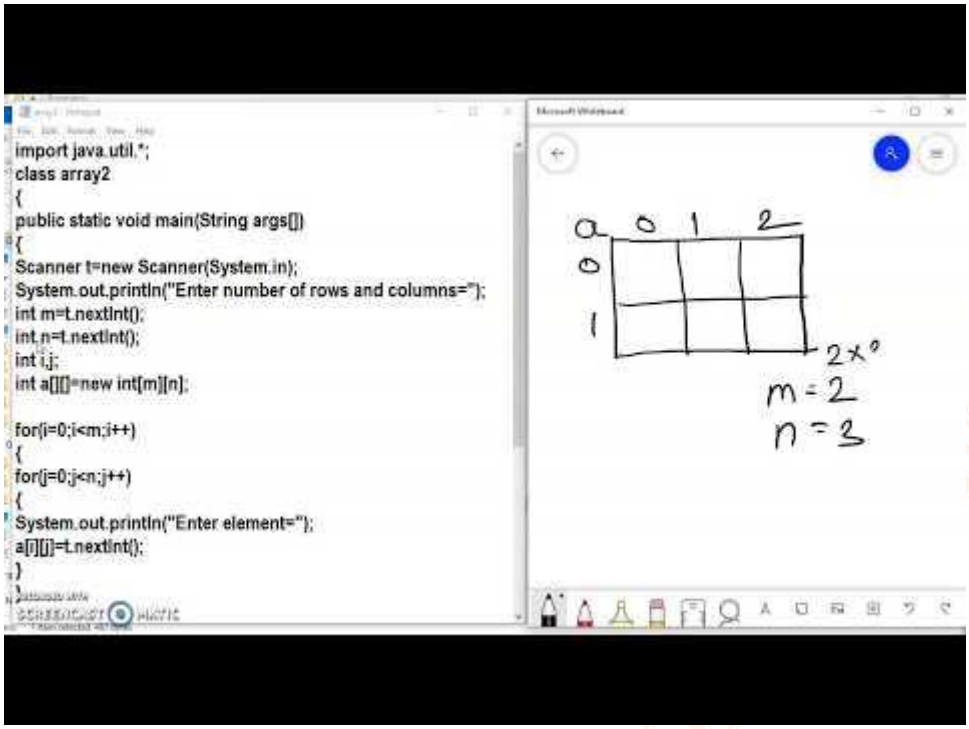
```
class Testarray3
{
public static void main(String args[])
{
//declaring and initializing 2D array
int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
//printing 2D array
for(int i=0;i<3;i++)
{
 for(int j=0;j<3;j++)
{
   System.out.print(arr[i][j]+" ");
 }
 System.out.println();
}
}
}
```

Output:

```
1 2 3
2 4 5
4 4 5
```

Video Link:





Web Links:

1. https://www.geeksforgeeks.org/multidimensional-arrays-in-java/
2. https://www.javatpoint.com/array-in-java

Question 5: Write a program to perform matrix addition in java

Answer:

```
import java.io.*;
class GFG {
    // Function to print Matrix
    static void printMatrix(int M[][],
                            int rowSize,
                            int colSize)
    {
        for (int i = 0; i < rowSize; i++) {
            for (int j = 0; j < colSize; j++)
                System.out.print(M[i][j] + " ");

            System.out.println();
        }
    }
    // Function to add the two matrices and store in matrix C
    static int[][] add(int A[][], int B[][],
                       int size)
    {
        int i, j;
```

```
        int C[][] = new int[size][size];
        for (i = 0; i < size; i++)
            for (j = 0; j < size; j++)
                C[i][j] = A[i][j] + B[i][j];
        return C;
    }
    public static void main(String[] args)
    {
        int size = 4;
        int A[][] = { { 1, 1, 1, 1 },
                      { 2, 2, 2, 2 },
                      { 3, 3, 3, 3 },
                      { 4, 4, 4, 4 } };
        // Print the matrices A
        System.out.println("\nMatrix A:");
        printMatrix(A, size, size);
        int B[][] = { { 1, 1, 1, 1 },
                      { 2, 2, 2, 2 },
                      { 3, 3, 3, 3 },
                      { 4, 4, 4, 4 } };
        // Print the matrices B
        System.out.println("\nMatrix B:");
        printMatrix(B, size, size);
        // Add the two matrices
        int C[][] = add(A, B, size);
        // Print the result
        System.out.println("\nResultant Matrix:");
        printMatrix(C, size, size);
    }
}
```

Output:

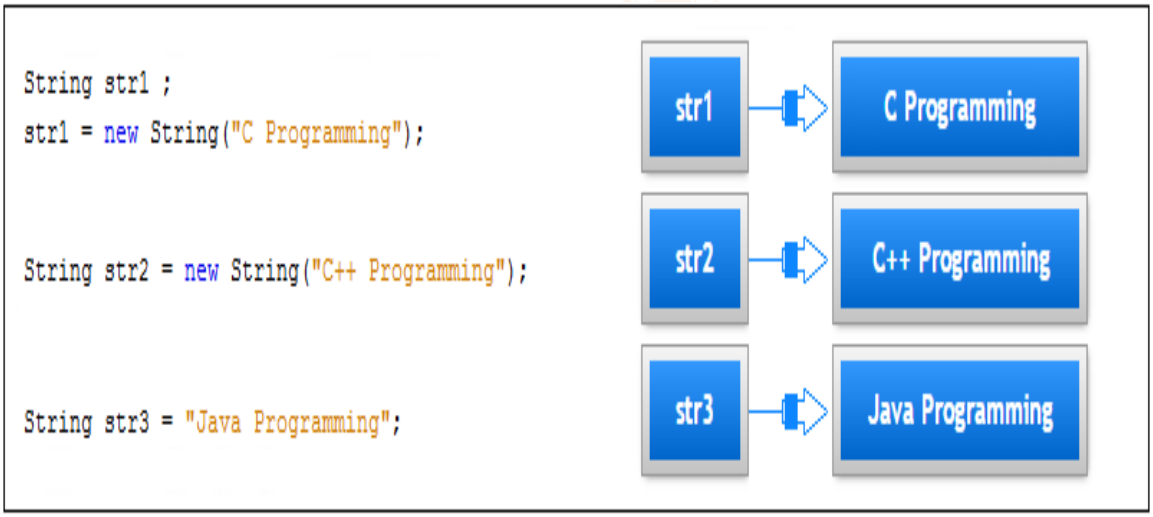
```
Matrix A:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4

Matrix B:
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

Resultant Matrix:
2 2 2 2
4 4 4 4
6 6 6 6
8 8 8 8

3.2 Strings:

Question 1: Explain declaration and initialization of string in java
Answer:
The primitive data types are specified by the keywords int, double, etc. While these keywords start with a lowercase letter, the keyword String, which represents the string data type, starts with an uppercase letter. This is because of the fact that keyword String is the name of a predefined class. A Java string is an instantiated object of the String class.
A String variable is simply a variable that stores a reference to an object of the class String. You declare a String variable in much the same way as you define a variable of one of the basic types. You can also three way to initialize it in the declaration. For example :



This declares the variable str1, str2 and str3 as type String and initializes it with a reference to a String object encapsulating the specified string.

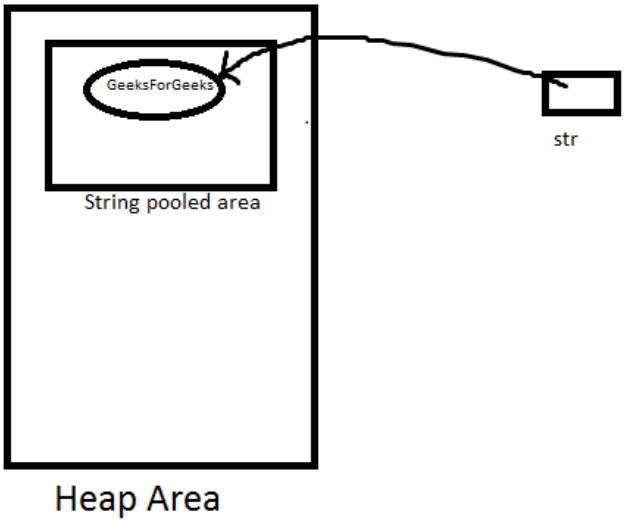
Initializing Strings in Java

1. Direct Initialization(String Constant) :
In this method, a String constant object will be created in String pooled area which is inside heap area in memory. As it is a constant, we can't modify it, i.e. String class is immutable.

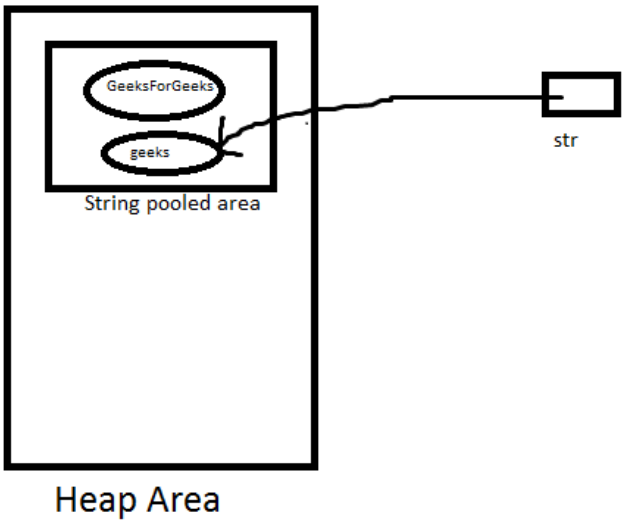
Examples:
String str = "GeeksForGeeks";

str = "geeks"; // This statement will make str
// point to new String constant("geeks")
// rather than modifying the previous
// String constant.

String str = “GeeksForGeeks”;

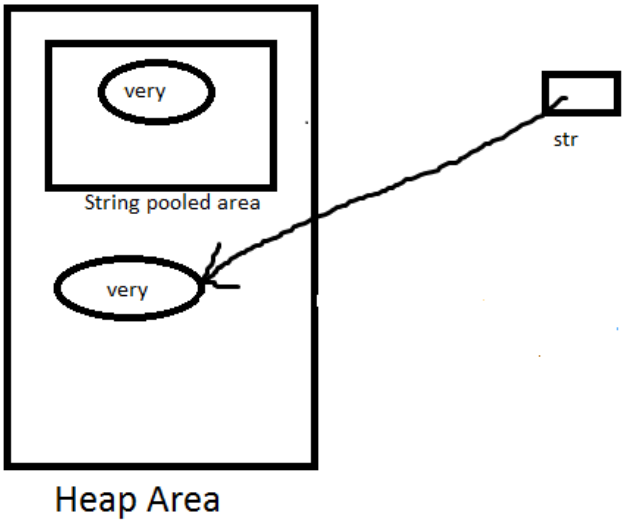


str = “geeks”;

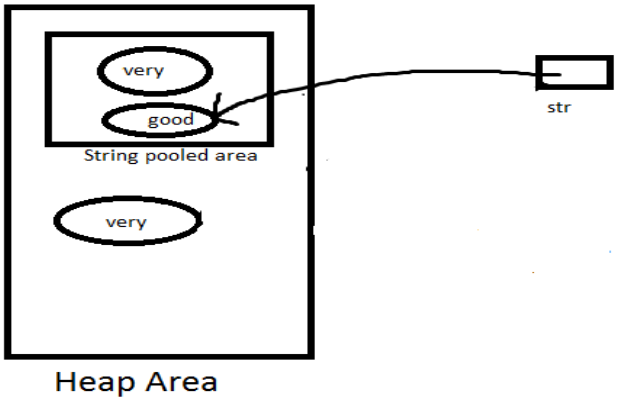


2. Object Initialization (Dynamic):
In this method, a String object will be created in heap area (not inside String pooled area as in upper case). We can modify it.Also with same value, a String constant is also created in String pooled area, but the variable will point to String object in heap area only.

Examples:
String str = new String("very");
str = "good";
String str = new String("very")



str = "good"
Now this is a direct assignment, so String constant with value "good" is created in String pooled area and str will point to that.



Question 2: Explain different string methods in java

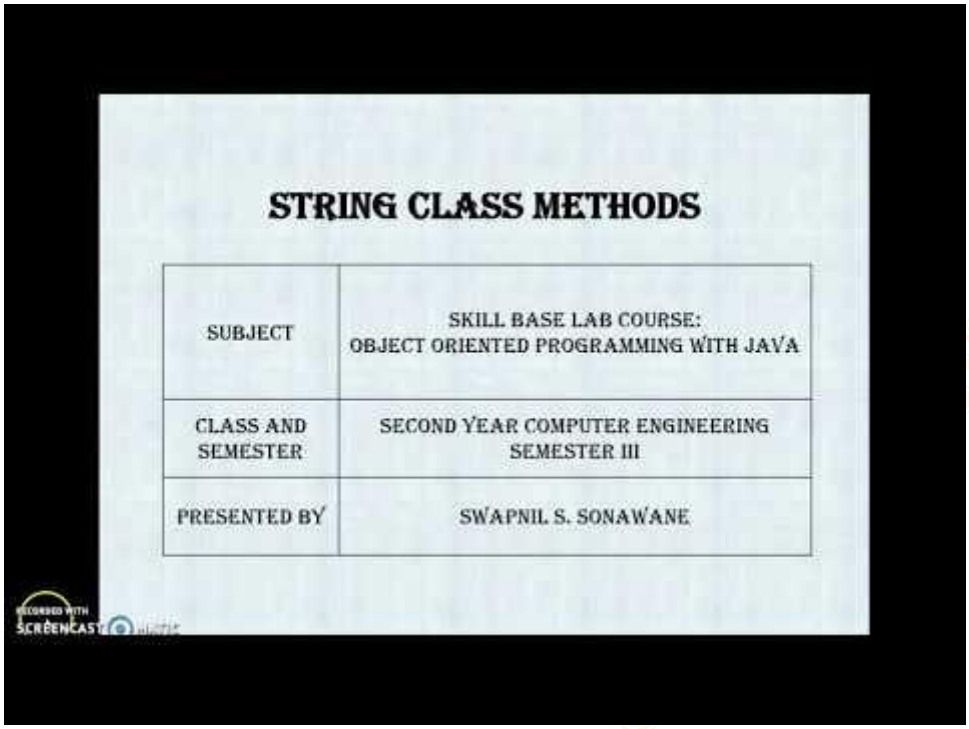
Answer:

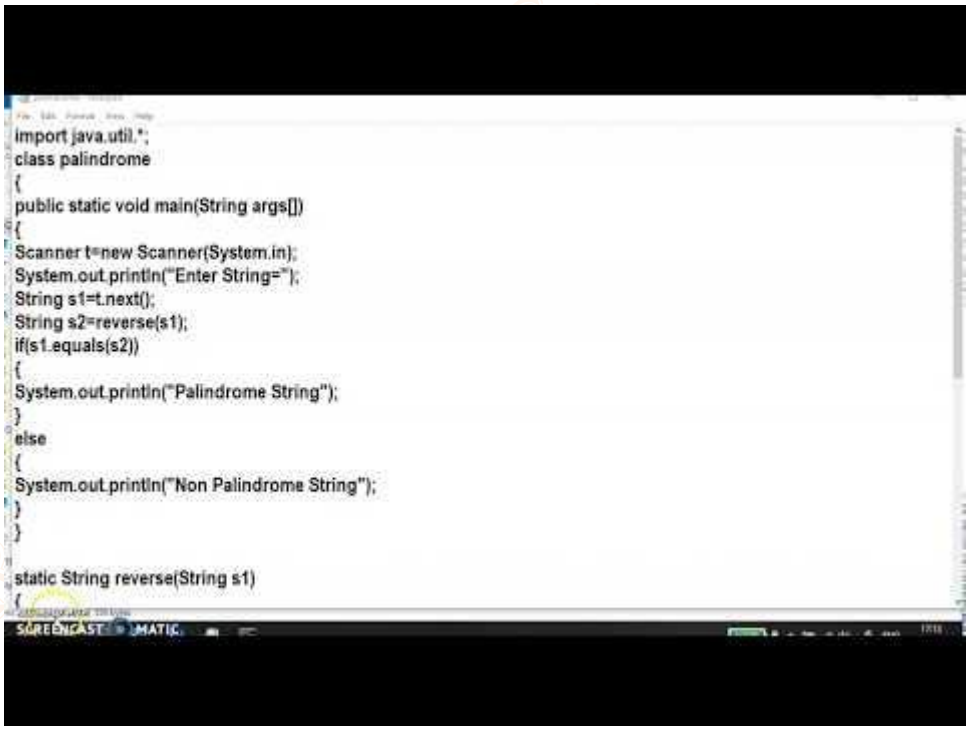
The String class has a set of built-in methods that you can use on strings.

Method	Description	Return Type
charAt()	Returns the character at the specified index (position)	char
compareTo()	Compares two strings lexicographically	int
compareToIgnoreCase()	Compares two strings lexicographically, ignoring case differences	int
concat()	Appends a string to the end of another string	String
contains()	Checks whether a string contains a sequence of characters	boolean
equals()	Compares two strings. Returns true if the strings are equal, and false if not	boolean
equalsIgnoreCase()	Compares two strings, ignoring case considerations	boolean
format()	Returns a formatted string using the specified locale, format string, and arguments	String
getBytes()	Encodes this String into a sequence of bytes using the named charset, storing the result into a new byte array	byte[]
getChars()	Copies characters from a string to an array of chars	void
indexOf()	Returns the position of the first found occurrence of specified characters in a string	int
intern()	Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index	String
isEmpty()	Checks whether a string is empty or not	boolean

lastIndexOf()	Returns the position of the last found occurrence of specified characters in a string	int
length()	Returns the length of a specified string	int
replace()	Searches a string for a specified value, and returns a new string where the specified values are replaced	String
replaceFirst()	Replaces the first occurrence of a substring that matches the given regular expression with the given replacement	String
replaceAll()	Replaces each substring of this string that matches the given regular expression with the given replacement	String
split()	Splits a string into an array of substrings	String[]
substring()	Extracts the characters from a string, beginning at a specified start position, and through the specified number of character	String
toCharArray()	Converts this string to a new character array	char[]
toLowerCase()	Converts a string to lower case letters	String
toString()	Returns the value of a String object	String
toUpperCase()	Converts a string to upper case letters	String
trim()	Removes whitespace from both ends of a string	String
valueOf()	Returns the primitive value of a String object	String

Video Link:





```
import java.util.*;
class palindrome
{
public static void main(String args[])
{
Scanner t=new Scanner(System.in);
System.out.println("Enter String=");
String s1=t.next();
String s2=reverse(s1);
if(s1.equals(s2))
{
System.out.println("Palindrome String");
}
else
{
System.out.println("Non Palindrome String");
}
}

static String reverse(String s1)
{
```

Web Links:

1. https://www.w3schools.com/java/java_ref_string.asp
2. https://www.javatpoint.com/java-string

Question 3: Write a program to count the occurrence of one string in another string

Answer:

```
class GFG
{
    static int countFreq(String pat, String txt)
   {
        int M = pat.length();
        int N = txt.length();
        int res = 0;
        /* A loop to slide pat[] one by one */
        for (int i = 0; i <= N - M; i++) {
            /* For current index i, check for
        pattern match */
            int j;
            for (j = 0; j < M; j++) {
                if (txt.charAt(i + j) != pat.charAt(j)) {
                    break;
                }
            }
            // if pat[0...M-1] = txt[i, i+1, ...i+M-1]
            if (j == M) {
                res++;
                j = 0;
            }
        }
        return res;
    }
    /* Driver program to test above function */
    static public void main(String[] args) {
        String txt = "mathematics";
        String pat = "mat";
        System.out.println(countFreq(pat, txt));
    }
}
```

Output :

2

3.3 StringBuffer Class:

Question 1: Explain StringBuffer class in java

Answer:

StringBuffer is a peer class of String that provides much of the functionality of strings. String represents fixed-length, immutable character sequences while StringBuffer represents mutable, growable and writable character sequences.

StringBuffer may have characters and substrings inserted in the middle or appended to the end. It will automatically grow to make room for such additions and often has more characters preallocated than are actually needed, to allow room for growth.

Example:

```
class GFG {
    public static void main(String[] args) {
        StringBuffer s = new StringBuffer("Geeksfor");
        s.append("Geeks");
        System.out.println(s); // returns GeeksforGeeks
        s.append(1);
        System.out.println(s); // returns GeeksforGeeks1
    } }
```

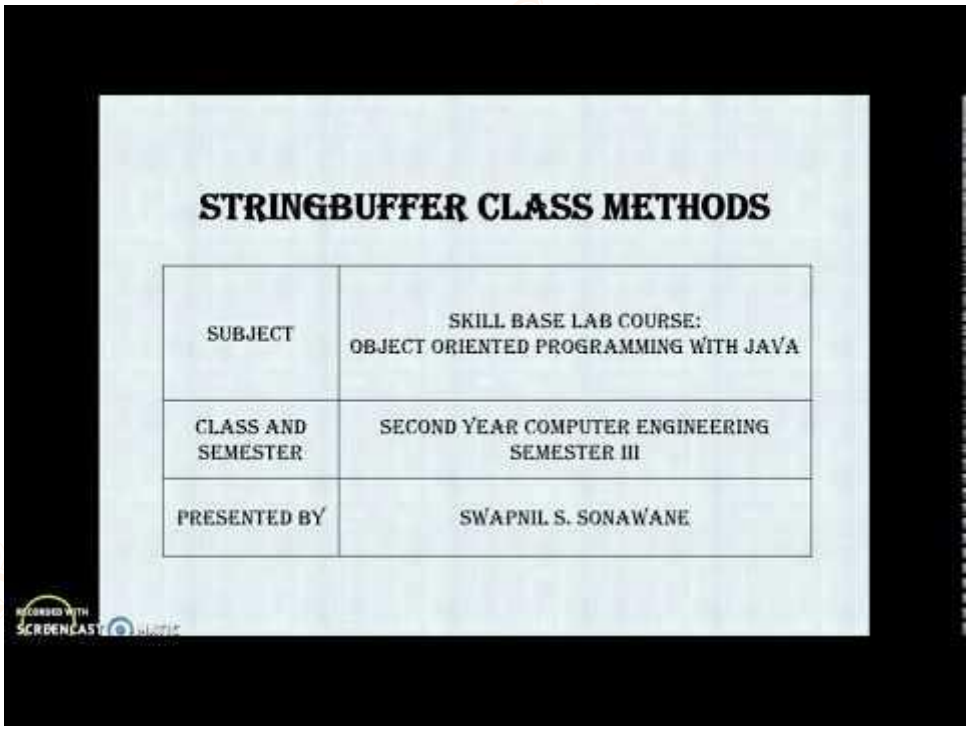
Output:

GeeksforGeeks
GeeksforGeeks1

Methods of StringBuffer class:

Method	Description
append(String s)	It is used to append the specified string with this string. The append() method is overloaded like append(char), append(boolean), append(int), append(float), append(double) etc.
insert(int offset, String s)	is used to insert the specified string with this string at the specified position. The insert() method is overloaded like insert(int, char), insert(int, boolean), insert(int, int), insert(int, float), insert(int, double) etc.
replace(int startIndex, int endIndex, String str)	is used to replace the string from specified startIndex and endIndex.
delete(int startIndex, int endIndex)	is used to delete the string from specified startIndex and endIndex.

Video Link:



Web Links:

1. https://www.javatpoint.com/StringBuffer-class
2. https://www.geeksforgeeks.org/stringbuffer-class-in-java/

3.4 Vectors:

Question 1: Explain Vector class in java

Answer:

Vector implements a dynamic array. It is similar to ArrayList, but with two differences-

- Vector is synchronized.
- Vector contains many legacy methods that are not part of the collections framework.

Vector proves to be very useful if you don't know the size of the array in advance or you just need one that can change sizes over the lifetime of a program.

Following is the list of constructors provided by the vector class.

Sr.No.	Constructor & Description
1	**Vector( )** This constructor creates a default vector, which has an initial size of 10.
2	**Vector(int size)** This constructor accepts an argument that equals to the required size, and creates a vector whose initial capacity is specified by size.
3	**Vector(int size, int incr)** This constructor creates a vector whose initial capacity is specified by size and whose increment is specified by incr. The increment specifies the number of elements to allocate each time that a vector is resized upward.

Vector class methods:

Sr. No	Methods	Description
1.	v.addElement(Object ob)	Used to add object at the end of vector
2.	v.insertElementAt(Object ob, int index)	Used to add object at a specific position
3.	v.removeElement(Object ob)	Removes the first (lowest-indexed) occurrence of the argument from this vector.
4.	v.removeElementAt(int index)	Removes object from specific index position

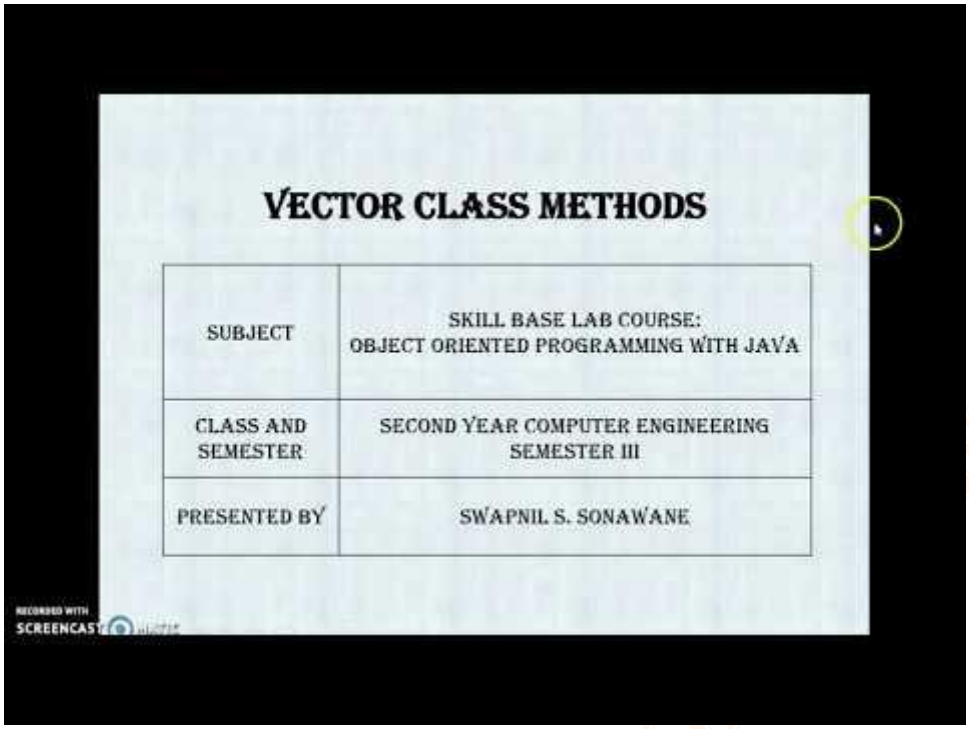
5.	v.elementAt(int index)	Retrieve object from specific index
6.	v.capacity()	Returns the current capacity of this vector
7.	v.size()	Returns the number of objects in this vector.

Example:

```
import java.util.*;
class vector
{
public static void main(String args[])
{
Scanner t=new Scanner(System.in);
Vector v=new Vector();
int n,c,i;
System.out.println("Enter number of students=");
n=t.nextInt();
for(i=0;i<n;i++)
{
System.out.println("Enter student name=");
String s1=t.next();
v.addElement(s1);
}
System.out.println("Menu\n1-Insert new student name\n2-Delete student
name\nEnter choice");
c=t.nextInt();
switch(c)
{
case 1:
{
System.out.println("Enter student name to be inserted and position=");
String s2=t.next();
int p=t.nextInt();
v.insertElementAt(s2,p);
for(i=0;i<v.size();i++)
{
System.out.println(v.elementAt(i).toString());
}
}
break;
case 2:
{
```

```
System.out.println("Enter student name to be deleted=");
String s2=t.next();
v.removeElement(s2);
for(i=0;i<v.size();i++)
{
System.out.println(v.elementAt(i).toString());
}
}
break;
default:
{
System.out.println("Invalid Choice");
}
}
}
}
```

Video Link:



Web Links:
1. https://www.tutorialspoint.com/java/java_vector_class.htm
2. https://www.javatpoint.com/java-vector

Question 2: Explain Wrapper class in java

Answer:

A Wrapper class is a class whose object wraps or contains a primitive data types. When we create an object to a wrapper class, it contains a field and in this field, we can store a primitive data types. In other words, we can wrap a primitive value into a wrapper class object.

Need of Wrapper Classes

1. They convert primitive data types into objects. Objects are needed if we wish to modify the arguments passed into a method (because primitive types are passed by value).
2. The classes in java.util package handles only objects and hence wrapper classes help in this case also.
3. Data structures in the Collection framework, such as ArrayList and Vector, store only objects (reference types) and not primitive types.
4. An object is needed to support synchronization in multithreading.

Primitive Data types and their Corresponding Wrapper class

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean

Example:

```
class WrapperExample1
{
public static void main(String args[])
{
//Converting int into Integer
int a=20;
Integer i=Integer.valueOf(a);//converting int into Integer explicitly
Integer j=a;//autoboxing, now compiler will write Integer.valueOf(a) internally
System.out.println(a+" "+i+" "+j);
}}
```

Output:

20 20 20

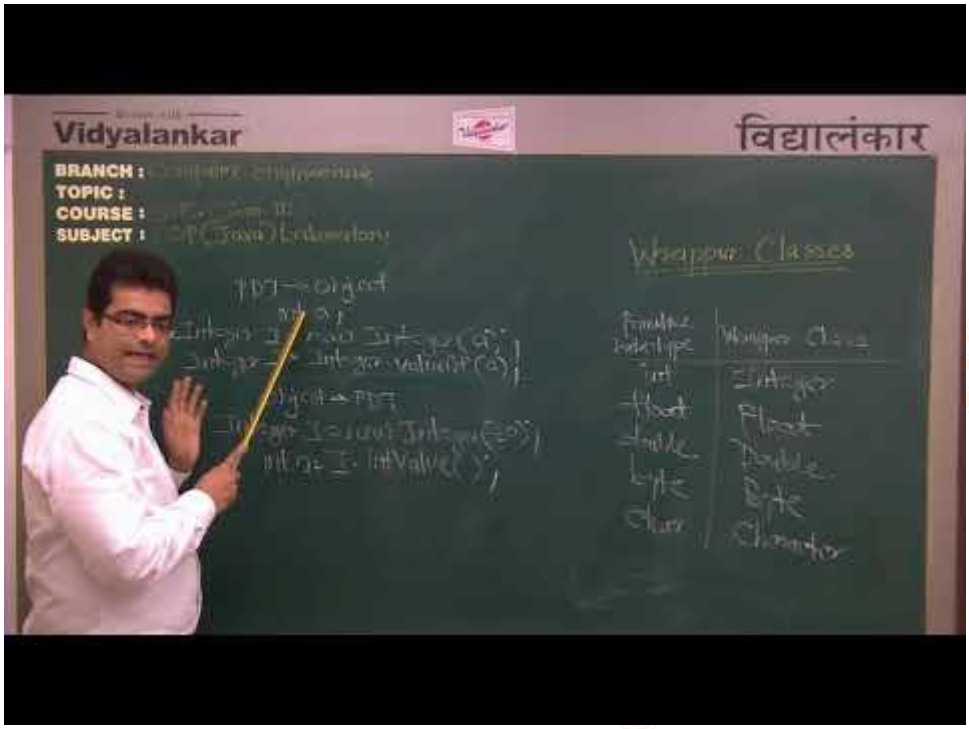
Example:

```
class WrapperExample2
{
public static void main(String args[])
{
//Converting Integer to int
Integer a=new Integer(3);
int i=a.intValue();//converting Integer to int explicitly
int j=a;//unboxing, now compiler will write a.intValue() internally
System.out.println(a+" "+i+" "+j);
}
}
```

Output:

3 3 3

Video Link:



Web Links:

1. https://www.geeksforgeeks.org/wrapper-classes java
2. https://www.javatpoint.com/wrapper-class-in-java

MODULE 4

Inheritance

Types of Inheritance

Method Overriding

Abstract class and Method

Final

Interface

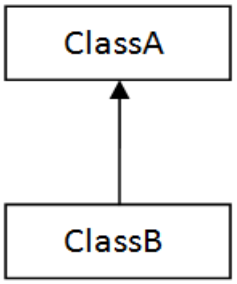
Module4: Inheritance

4.1 Types of Inheritance:

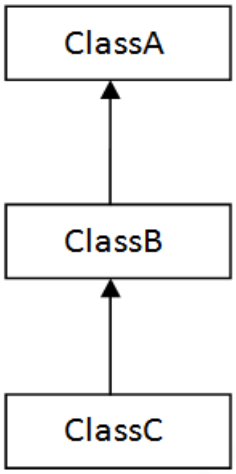
Question 1: Explain different types of inheritance

Answer:

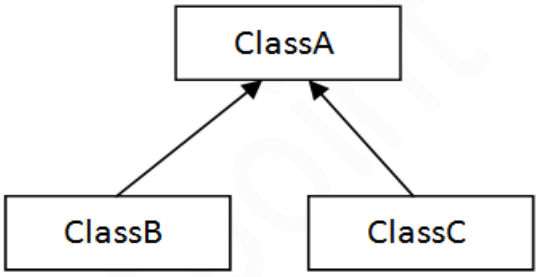
On the basis of class, there can be three types of inheritance in java: single, multilevel and hierarchical.In java programming, multiple and hybrid inheritance is supported through interface only. We will learn about interfaces later.



1) Single

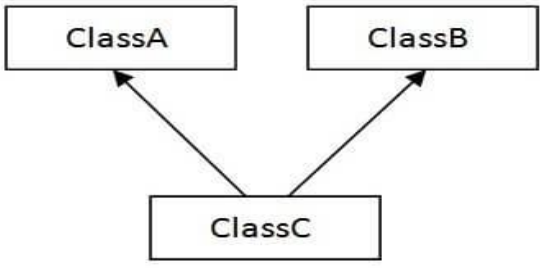


2) Multilevel

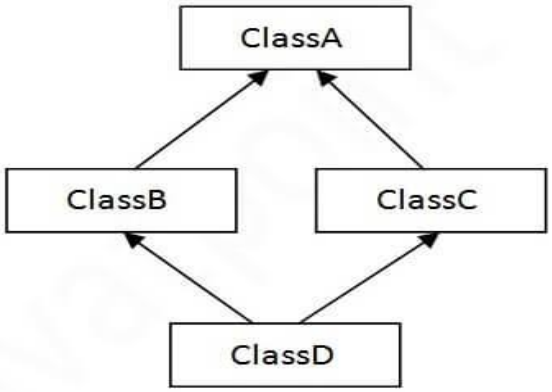


3) Hierarchical

When one class inherits multiple classes, it is known as multiple inheritance. For Example:



4) Multiple



5) Hybrid

.

Single Inheritance Example

When a class inherits another class, it is known as a *single inheritance*. In the example given below, Dog class inherits the Animal class, so there is the single inheritance.

Example:

```
class Animal
{
void eat(){System.out.println("eating...");
}
}
class Dog extends Animal
{
void bark(){System.out.println("barking...");
}
}
class TestInheritance
{
public static void main(String args[])
{
Dog d=new Dog();
d.bark();
d.eat();
}
}
```

Output:

```
barking...
eating...
```

Multilevel Inheritance Example

When there is a chain of inheritance, it is known as *multilevel inheritance*. As you can see in the example given below, BabyDog class inherits the Dog class which again inherits the Animal class, so there is a multilevel inheritance.

Example:

```
class Animal
{
void eat(){System.out.println("eating...");
}
}
```

```
class Dog extends Animal
{
void bark(){System.out.println("barking...");
}
}
class BabyDog extends Dog
{
void weep(){System.out.println("weeping...");
}
}
class TestInheritance2
{
public static void main(String args[])
{
BabyDog d=new BabyDog();
d.weep();
d.bark();
d.eat();
}
}
```

Output:

```
weeping...
barking...
eating...
```

Hierarchical Inheritance Example

When two or more classes inherits a single class, it is known as *hierarchical inheritance*. In the example given below, Dog and Cat classes inherits the Animal class, so there is hierarchical inheritance.

Example:

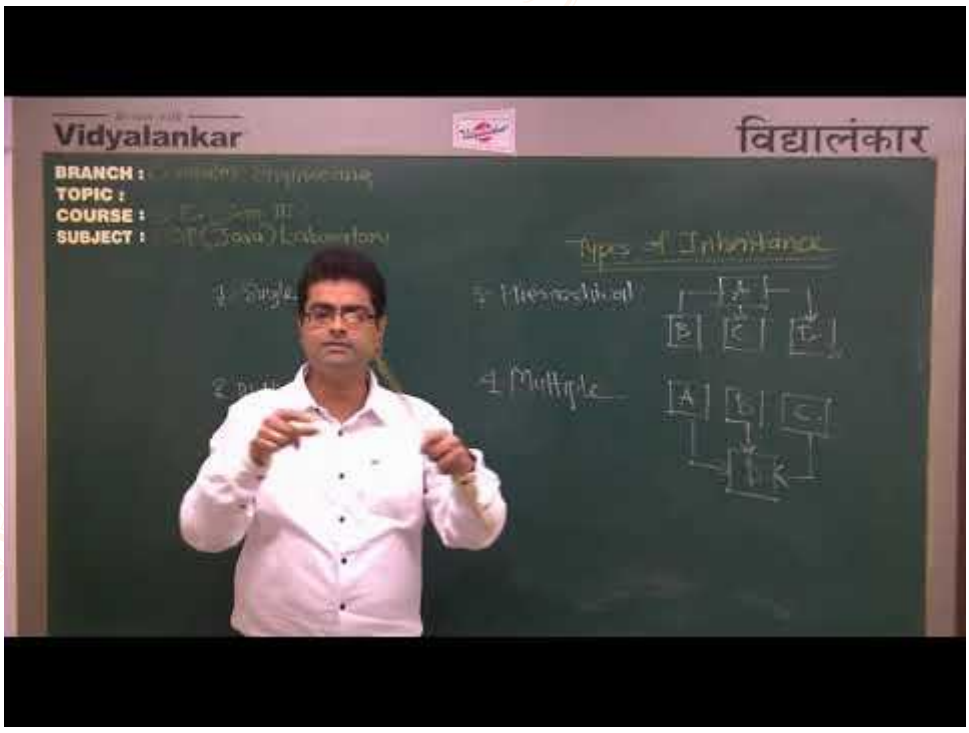
```
class Animal
{
void eat(){System.out.println("eating...");
}
}
class Dog extends Animal
{
void bark(){System.out.println("barking...");
}
}
class Cat extends Animal
```

```
{
void meow(){System.out.println("meowing...");
}
}
class TestInheritance3
{
public static void main(String args[])
{
Cat c=new Cat();
c.meow();
c.eat();
//c.bark();//C.T.Error
}
}
```

Output:

```
meowing...
eating...
```

Video Link:



Web Links:

1. https://www.javatpoint.com/inheritance-in-java

4.2 Method Overriding:

Question 1: Explain method overriding with example
Answer:
If subclass (child class) has the same method as declared in the parent class, it is known as method overriding in Java.
In other words, If a subclass provides the specific implementation of the method that has been declared by one of its parent class, it is known as method overriding.

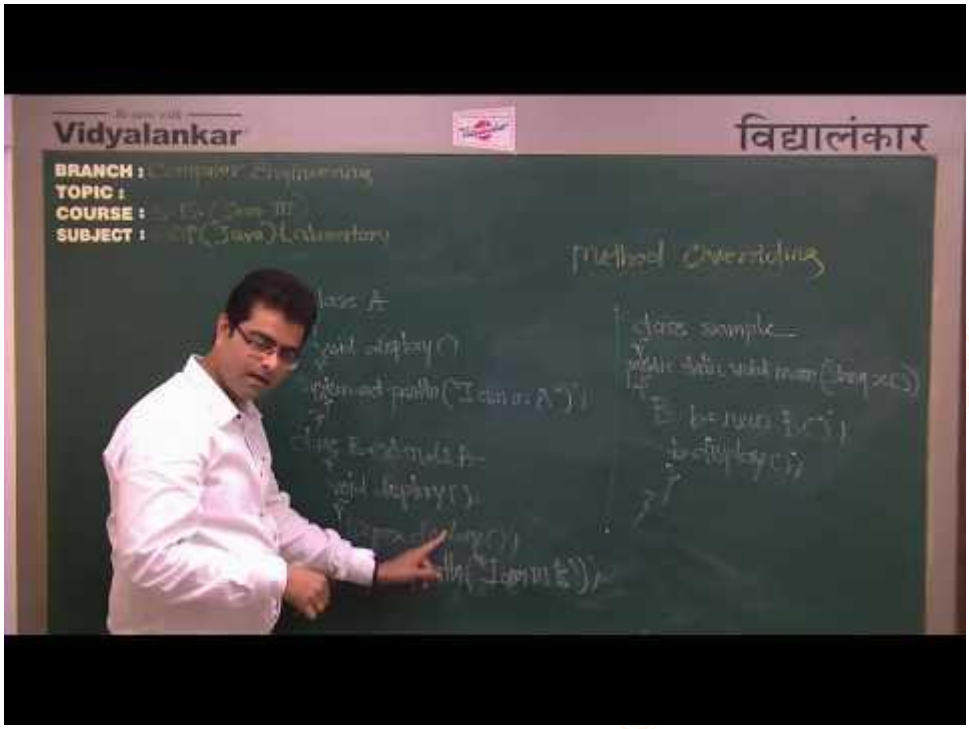
Usage of Java Method Overriding
Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
Method overriding is used for runtime polymorphism

Rules for Java Method Overriding
The method must have the same name as in the parent class
The method must have the same parameter as in the parent class.

Example:
```
class Vehicle
{
void run()
{
System.out.println("Vehicle is running");
}
}
//Creating a child class
class Bike2 extends Vehicle
{
void run()
{
System.out.println("Bike is running safely");
}
public static void main(String args[])
{
  Bike2 obj = new Bike2();//creating object
  obj.run();//calling method
  }
}
```

Output:
Bike is running safely

Video Link:



Web Links:
1. https://www.javatpoint.com/method-overriding-in-java

Question 2: Explain use of "super" keyword with example
Answer:
The super keyword in Java is a reference variable which is used to refer immediate parent class object.
Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.

Usage of Java super Keyword

- super can be used to refer immediate parent class instance variable.
- super can be used to invoke immediate parent class method.
- super() can be used to invoke immediate parent class constructor.

super can be used to invoke parent class method
The super keyword can also be used to invoke parent class method. It should be used if subclass contains the same method as parent class. In other words, it is used if method is overridden.

Example:

```
class Animal
{
void eat(){System.out.println("eating...");
}
}
class Dog extends Animal{
void eat(){System.out.println("eating bread...");
}
void bark(){System.out.println("barking...");
}
void work()
{
super.eat();
bark();
}
}
class TestSuper2
{
public static void main(String args[])
{
Dog d=new Dog();
d.work();
}
}
```

Output:

```
eating...
barking...
```

super is used to invoke parent class constructor.

The super keyword can also be used to invoke the parent class constructor.

Example:

```
class Animal
{
Animal(){System.out.println("animal is created");
}
}
class Dog extends Animal
{
Dog(){
super();
```

```
System.out.println("dog is created");
}
}
class TestSuper3
{
public static void main(String args[])
{
Dog d=new Dog();
}
}
```

Output:
animal is created
dog is created

Video Link:

Web Links:
1. https://www.javatpoint.com/super-keyword
2. https://www.geeksforgeeks.org/super-keyword/

4.3 Abstract Class and Methods:

Question 1: Explain abstract method in java

Answer:

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.

- abstract keyword is used to declare the method as abstract.
- You have to place the abstract keyword before the method name in the method declaration.
- An abstract method contains a method signature, but no method body.
- Instead of curly braces, an abstract method will have a semoi colon (;) at the end.

Following is an example of the abstract method.

Example:

```
public abstract class Employee
{
   private String name;
   private String address;
   private int number;

   public abstract double computePay();
   // Remainder of class definition
}
```

Declaring a method as abstract has two consequences –

- The class containing it must be declared as abstract.
- Any class inheriting the current class must either override the abstract method or declare itself as abstract.

Example:

```
abstract class Bike
{
  abstract void run();
}
class Honda4 extends Bike
{
void run(){System.out.println("running safely");
}
public static void main(String args[])
{
```

```
 Bike obj = new Honda4();
 obj.run();
}
}
```

Output:
running safely

Video Link:

Web Links:
https://www.javatpoint.com/abstract-class-in-java

4.3 final Keyword:

Question 1: Explain use of final keyword in java
Answer:
The final keyword in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

- variable
- method
- class

The final keyword can be applied with the variables, a final variable that have no value it is called blank final variable or uninitialized final variable. It can be initialized in the constructor only. The blank final variable can be static also which will be initialized in

the static block only. We will have detailed learning of these. Let's first learn the basics of final keyword.

final keyword in java

1) Java final variable

If you make any variable as final, you cannot change the value of final variable(It will be constant).

Example:

```
class Bike9
{
 final int speedlimit=90;//final variable
 void run(){
  speedlimit=400;
 }
 public static void main(String args[])
{
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Output:

Compile Time Error

2) Java final method

If you make any method as final, you cannot override it.

Example:

```
class Bike
{
  final void run(){System.out.println("running");
}  }
class Honda extends Bike
{
   void run(){System.out.println("running safely with 100kmph");
}
   public static void main(String args[])
{
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:
Compile Time Error

3) Java final class

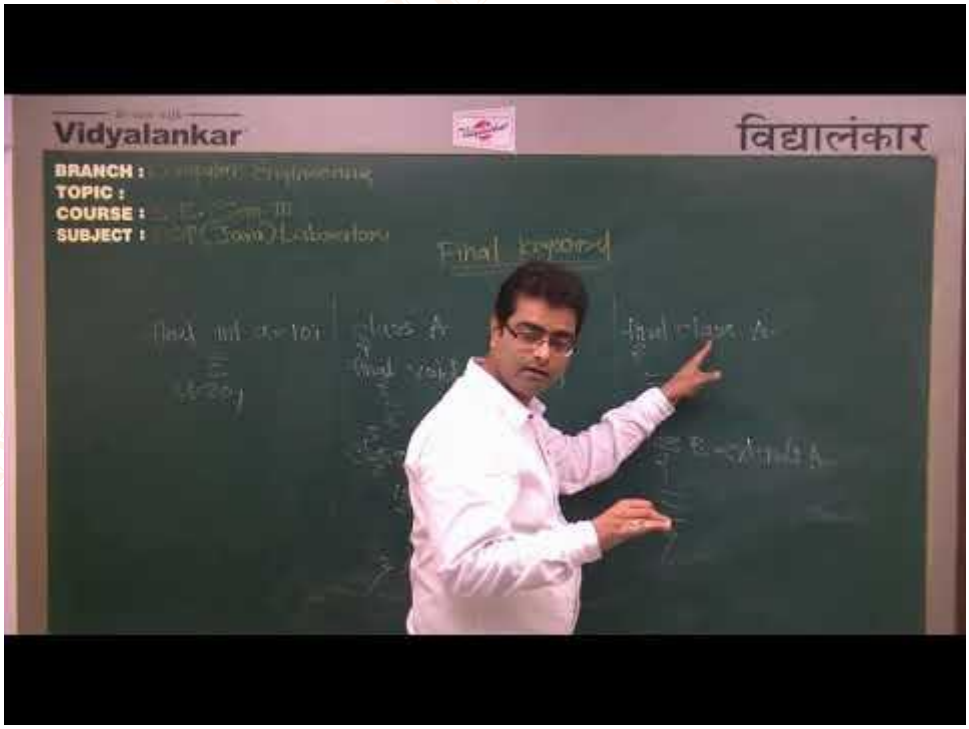
If you make any class as final, you cannot extend it.

Example:

```
final class Bike{}
class Honda1 extends Bike
{
  void run(){System.out.println("running safely with 100kmph");
}
  public static void main(String args[])
{
  Honda1 honda= new Honda1();
  honda.run();
  } }
```

Output:
Compile Time Error

Video Link:



Web Links:
1. https://www.javatpoint.com/final-keyword

4.3 Interface:

Question 1: Explain interface with example
Answer:
An interface is a completely "abstract class" that is used to group related methods with empty bodies:

Example:
```
// interface
interface Animal
{
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" (kinda like inherited) by another class with the implements keyword (instead of extends). The body of the interface method is provided by the "implement" class:

Example:
```
// Interface
interface Animal
{
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}
// Pig "implements" the Animal interface
class Pig implements Animal
{
  public void animalSound()
  {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep()
  {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}
class MyMainClass
{
  public static void main(String[] args)
```

```
{
   Pig myPig = new Pig();  // Create a Pig object
   myPig.animalSound();
   myPig.sleep();
  }
}
```

Notes on Interfaces:

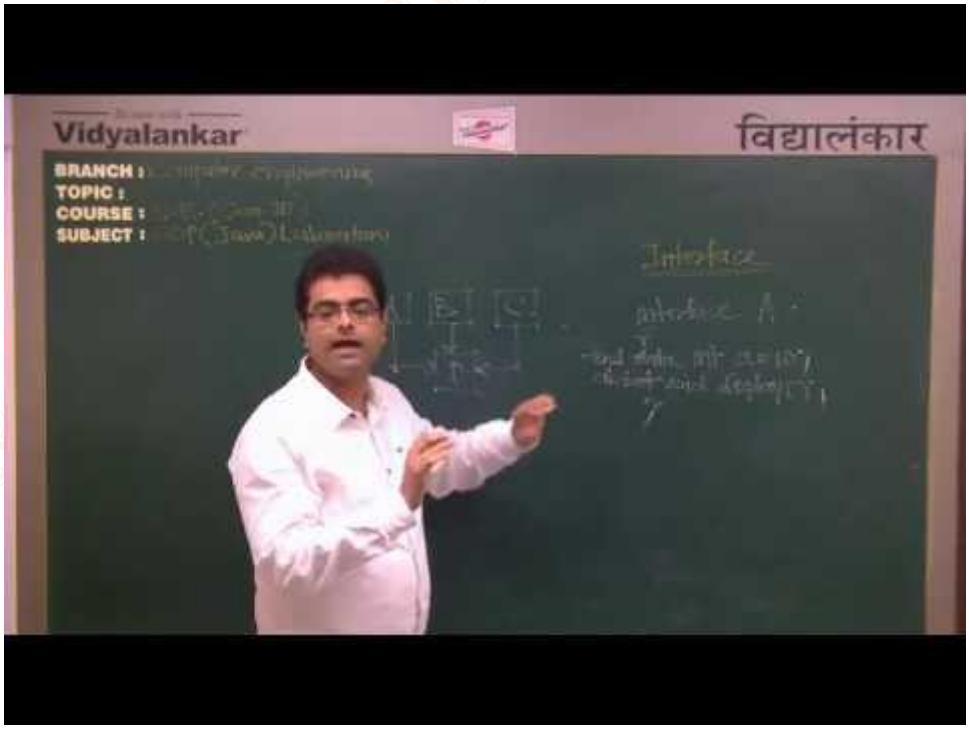
- Like abstract classes, interfaces cannot be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final
- An interface cannot contain a constructor (as it cannot be used to create objects)

Example:

```
import java.util.*;
interface account
{
int accno=1;
void show();
}
class person
{
String name;
void accept(){
Scanner t=new Scanner(System.in);
System.out.println("Enter name ");
name=t.next();
}
void display()
{
System.out.println("Name="+name);
}}
class customer extends person implements account
{
double balance;
void accept(){
super.accept();
Scanner t=new Scanner(System.in);
```

```
System.out.println("Enter balance ");
balance=t.nextDouble();
}
void display()
{
super.display();
System.out.println("Balance="+balance);
}
public void show()
{
System.out.println("Account number="+accno);
}}
class inter{
public static void main(String x[])
{
customer c=new customer();
c.accept();
c.show();
c.display();
}}
```

Video Link:



MODULE 5

Exception Handling and Multithreading

Exception Handling

User Defined Exceptions

Thread Lifecycle

Thread Class Methods

Creating Threads

Module 5: Exception Handling and Multithreading

5.1 Exception Handling:

Question 1: Explain try and catch in exception handling
Answer:
try block:
The try block contains set of statements where an exception can occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must be followed by catch blocks or finally block or both.

Syntax of try block

```
try
{
  //statements that may cause an exception
}
```

While writing a program, if you think that certain statements in a program can throw a exception, enclosed them in try block and handle that exception

catch block:
A catch block is where you handle the exceptions, this block must follow the try block. A single try block can have several catch blocks associated with it. You can catch different exceptions in different catch blocks. When an exception occurs in try block, the corresponding catch block that handles that particular exception executes. For example if an arithmetic exception occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

Syntax of try catch in java

```
try
{
    //statements that may cause an exception
}
catch (exception(type) e(object))
{
    //error handling code
}
```

Example:
If an exception occurs in try block then the control of execution is passed to the corresponding catch block. A single try block can have multiple catch blocks associated with it, you should place the catch blocks in such a way that the generic exception handler catch block is at the last(see in the example below).

The generic exception handler can handle all the exceptions but you should place is at the end, if you place it at the before all the catch blocks then it will display the generic message. You always want to give the user a meaningful message for each type of exception rather then a generic message.

```
class Example1
{
   public static void main(String args[])
    {
     int num1, num2;
     try {
        /* We suspect that this block of statement can throw
         * exception so we handled it by placing these statements
         * inside try and handled the exception in catch block
         */
        num1 = 0;
        num2 = 62 / num1;
        System.out.println(num2);
        System.out.println("Hey I'm at the end of try block");
     }
     catch (ArithmeticException e)
    {
        /* This block will only execute if any Arithmetic exception
         * occurs in try block
         */
        System.out.println("You should not divide a number by zero");
     }
     catch (Exception e)
      {
        /* This is a generic Exception handler which means it can handle
         * all the exceptions. This will execute if the exception is not
         * handled by previous catch blocks.
         */
        System.out.println("Exception occurred");
     }
     System.out.println("I'm out of try-catch block in Java.");
   }
}
```

Output:

```
You should not divide a number by zero
I'm out of try-catch block in Java.
```

Video Link:

Web Links:
1. https://beginnersbook.com/2013/04/try-catch-in-java/
2. https://www.w3schools.com/java/java_try_catch.asp

Question 2: Write a program to demonstrate the use of multiple catch statement
Answer:

```
import java.util.*;
class mulcatch
{
public static void main(String args[])
{
try
{
Scanner t=new Scanner(System.in);
System.out.println("Enter 2 numbers=");
int a=t.nextInt();
int b=t.nextInt();
int c=a/b;
System.out.println("Division="+c);
}
catch(ArithmeticException ae)
{
System.out.println("Arithmetic Exception Occurs");
```

```
}
catch(InputMismatchException ie)
{
System.out.println("Input Mismatch Exception Occurs");
}
}
}
```

Question 3: Explain “finally” in exception handling

Answer:

Java finally block is a block that is used to execute important code such as closing connection, stream etc.
Java finally block is always executed whether exception is handled or not.
Java finally block follows try or catch block.
Finally block in java can be used to put "cleanup" code such as closing a file, closing connection etc.

Usage of Java finally

Let's see the different cases where java finally block can be used.

Case 1

Let's see the java finally example where exception doesn't occur.

```
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 the code...");
  }
}
```

Output:

```
5
finally block is always executed
rest of the code...
```

Case 2

Let's see the java finally example where exception occurs and not handled.

```
class TestFinallyBlock1{
  public static void main(String args[]){
```

```
  try{
   int data=25/0;
   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 the code...");
  }
}
```

Output:

```
finally block is always executed
Exception in thread main java.lang.ArithmeticException:/ by zero
```

Case 3

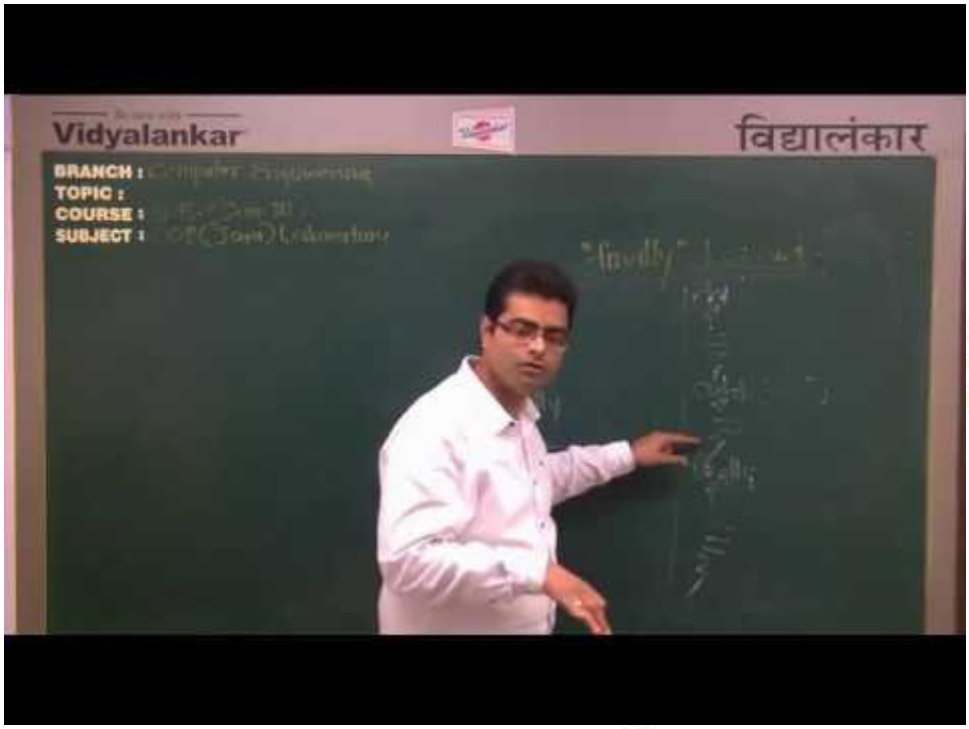
Let's see the java finally example where exception occurs and handled.

```
public class TestFinallyBlock2
  {
  public static void main(String args[])
  {
  try
  {
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){System.out.println(e);}
  finally{System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Output:

```
Exception in thread main java.lang.ArithmeticException:/ by zero
finally block is always executed
rest of the code...
```

Video Link:



Web Links:
https://www.javatpoint.com/finally-block-in-exception-handling

Question 4: Explain "throws" in exception handling
Answer:
The Java throws keyword is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.
Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Syntax of java throws
return_type method_name() throws exception_class_name
{
//method code
}

Which exception should be declared
-Checked exception only, because:

- Unchecked Exception: Under your control so correct your code.

- error: beyond your control e.g. you are unable to do anything if there occurs VirtualMachineError or StackOverflowError.

Example:
Let's see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

```
import java.io.IOException;
class Testthrows1
{
  void m()throws IOException
{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p()
{
   try
{
    n();
   }catch(Exception e){System.out.println("exception handled");
}
  }
  public static void main(String args[])
{
   Testthrows1 obj=new Testthrows1();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

Output:
exception handled
normal flow...

Video Link:

Web Links:
1. https://www.javatpoint.com/throws-keyword-and-difference-between-throw-and-throws

5.2 User Defined Exceptions:

Question 1: Explain "throw" in exception handling
Answer:
The Java throw keyword is used to explicitly throw an exception.
We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw custom exception. We will see custom exceptions later.

Example:

```
public class MyClass {
  static void checkAge(int age) {
    if (age < 18) {
      throw new ArithmeticException("Access denied - You must be at least 18 years old.");
    }
    else {
      System.out.println("Access granted - You are old enough!");
    }
```

```
  }
  public static void main(String[] args) {
    checkAge(15); // Set age to 15 (which is below 18...)
  }
}
```

Output:

Exception in thread "main" java.lang.ArithmeticException: Access denied - You must be at least 18 years old.
        at MyClass.checkAge(MyClass.java:4)
        at MyClass.main(MyClass.java:12)

Video Link:

Web Links:

1. https://www.javatpoint.com/throw-keyword
2. https://www.w3schools.com/java/ref_keyword_throw.asp

Question 2: Explain user defined exceptions in Java

Answer:

User Defined Exception or custom exception is creating your own exception class and throws that exception using 'throw' keyword. This can be done by extending the class Exception.

Example:

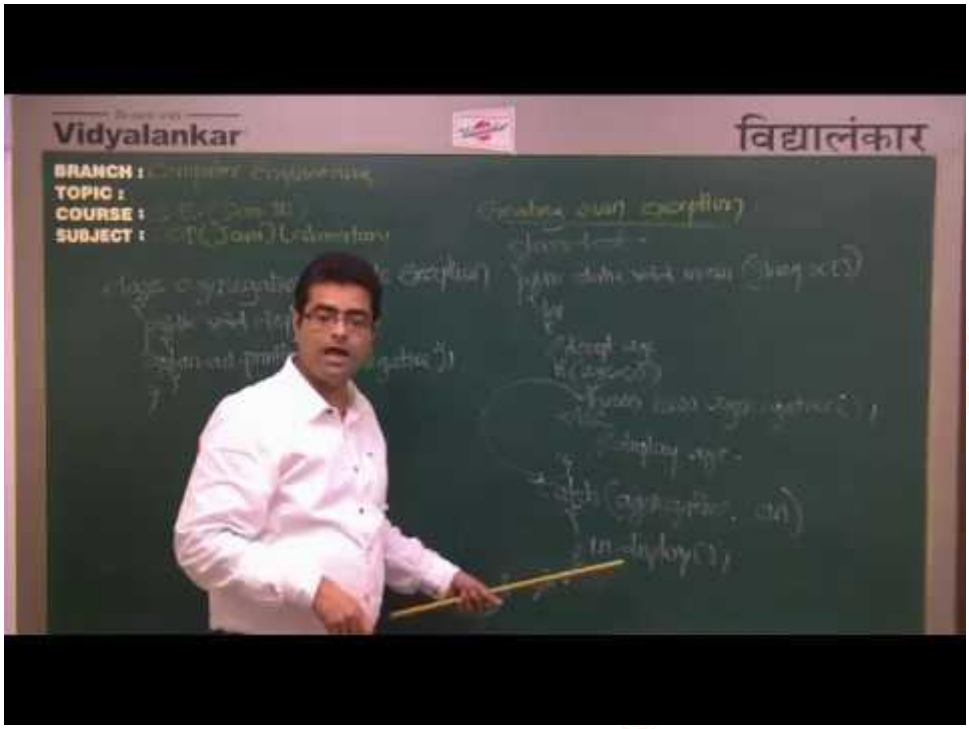
```
class MyException extends Exception{
   String str1;
   /* Constructor of custom exception class
    * here I am copying the message that we are passing while
    * throwing the exception to a string and then displaying
    * that string along with the message.
    */
   MyException(String str2)
  {
        str1=str2;
   }
   public String toString()
  {
        return ("MyException Occurred: "+str1) ;
   }
}

class Example1
{
   public static void main(String args[])
    {
        try
        {
              System.out.println("Starting of try block");
              // I'm throwing the custom exception using throw
              throw new MyException("This is My error Message");
        }
        catch(MyException exp)
        {
              System.out.println("Catch Block") ;
              System.out.println(exp) ;
        }
   }
}
```

Output:

```
Starting of try block
Catch Block
MyException Occurred: This is My error Message
```

Video Link:

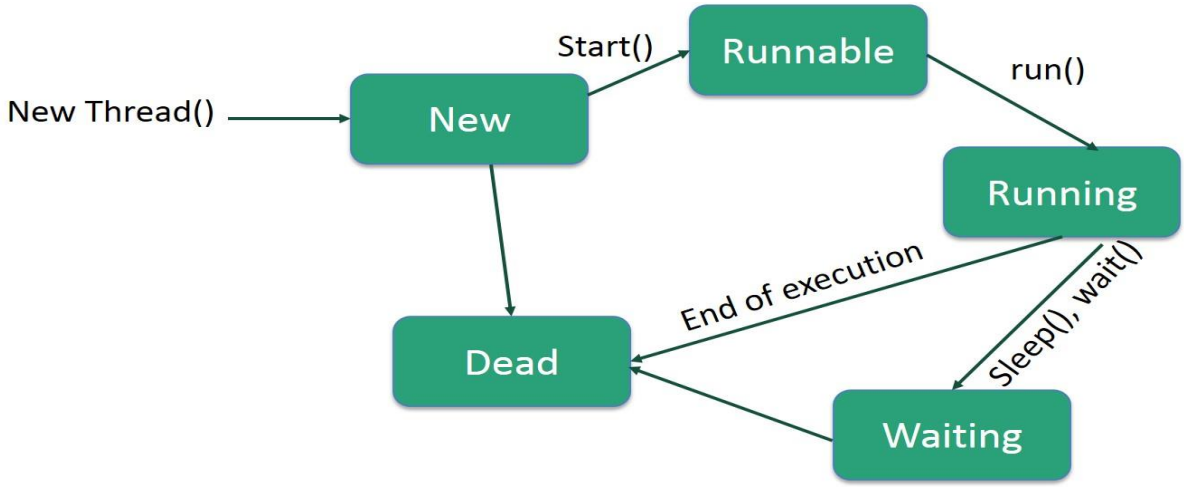


Web Links:
1. https://beginnersbook.com/2013/04/user-defined-exception-in-java/
2. https://www.guru99.com/java-user-defined-exception.html

5.3 Thread Life Cycle:

Question 1: Explain thread life cycle with diagram
Answer:
A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. The following diagram shows the complete life cycle of a thread.
Following are the stages of the life cycle −

- **New** − A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a **born thread**.
- **Runnable** − After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting** − Sometimes, a thread transitions to the waiting state while the thread waits for another thread to perform a task. A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.



- **Timed Waiting** − A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated (Dead)** − A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Video Link:

Web Links:

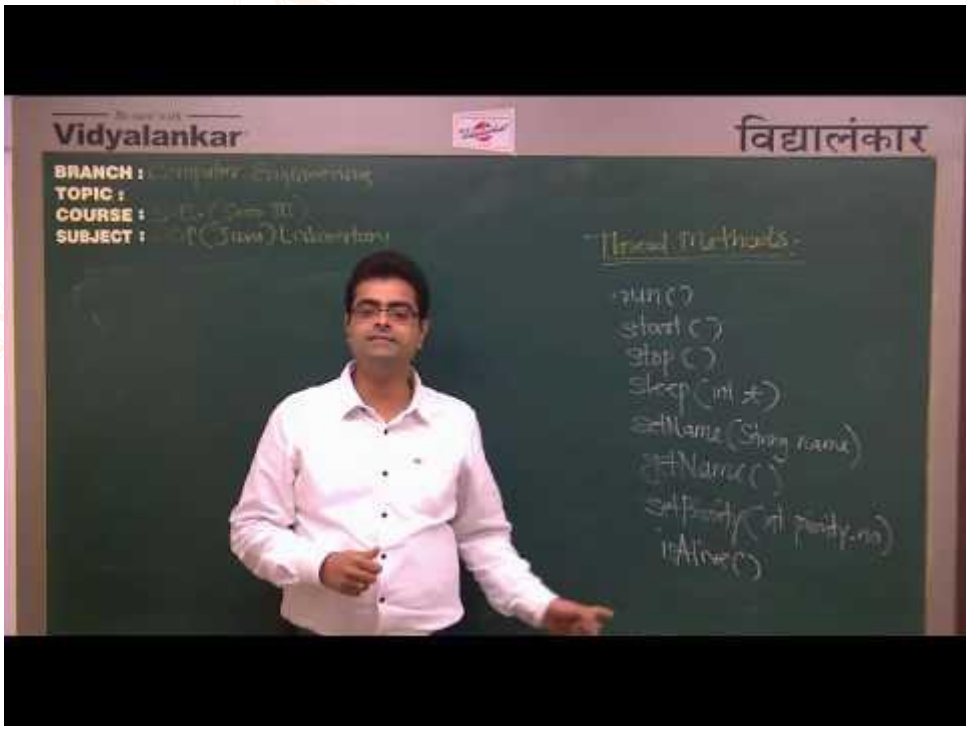
1. https://www.tutorialspoint.com/java/java_multithreading.htm

5.4 Thread Class Methods:

Question 1: Explain different Thread class methods
Answer:

Method	Meaning
getName()	Obtain thread's name
getPriority()	Obtain thread's priority
isAlive()	Determine if a thread is still running
join()	Wait for a thread to terminate
run()	Entry point for the thread
sleep()	Suspend a thread for a period of time
start()	Start a thread by calling its run method

Video Link:



Web Links:
1. https://www.javatpoint.com/creating-thread
2. https://dzone.com/articles/java-thread-tutorial-creating-threads-and-multithr

5.4 Creating Threads:

Question 1: Explain how to create a thread in java
Answer:
There are two ways to create a thread:

- By extending Thread class
- 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.

Runnable interface:
The Runnable interface should be implemented by any class whose instances are intended to be executed by a thread. Runnable interface have only one method named run().

1) Java Thread Example by extending Thread class

Example:

```
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

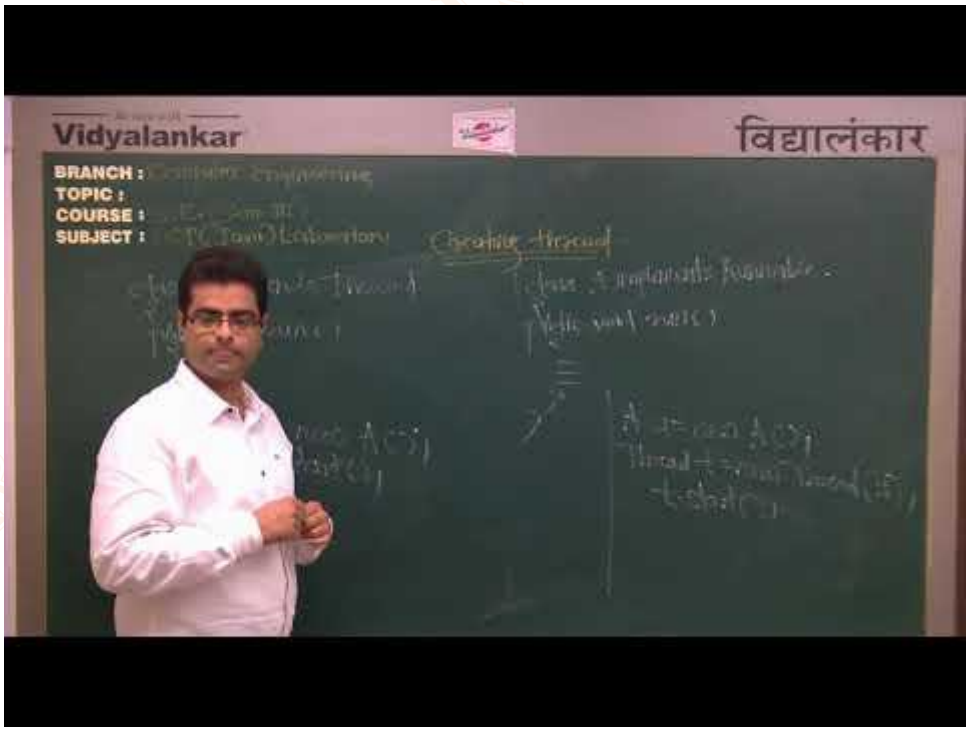
Example:

```
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);
t1.start();
 }
}
```

Output:

thread is running...

Video Link:



Web Links:

1. https://www.javatpoint.com/creating-thread

Question 2: Write a program to create two threads, one thread is used to print even numbers from 1 to 20 whereas other thread is used to print odd numbers from 1 to 20.

Answer:

```
class A extends Thread
{
public void run()
{
int i;
for(i=2;i<=20;i=i+2)
{
System.out.println("Even="+i);
}
}
}
class B extends Thread
{
public void run()
{
int i;
for(i=1;i<=20;i=i+2)
{
System.out.println("Odd="+i);
}
}
}
class MT1
{
public static void main(String x[])
{
A a=new A();
B b=new B();
a.start();
b.start();
}
}
```

Question 3: Write a program to create two threads, one thread is used to print five "*" whereas other thread is used to print five "#" and each symbol should print after some time sleep.

Answer:

```
class A extends Thread
{
```

```
public void run()
{
int i;
for(i=1;i<=5;i++)
{
System.out.println("*");
try
{
Thread.sleep(500);
}
catch(InterruptedException ie)
{
System.out.println("Exception Occurs");
}}}}

class B extends Thread
{
public void run()
{
int i;
for(i=1;i<=5;i++)
{
System.out.println("#");
try
{
Thread.sleep(500);
}
catch(InterruptedException ie)
{
System.out.println("Exception Occurs");
}
}
}}
class MT2
{
public static void main(String x[])
{
A a=new A();
B b=new B();
a.start();
b.start();
}
}
```

MODULE 6

GUI Programming in JAVA

Applet and Applet Lifecycle

Creating Applet

Parameter Passing to Applet

Font and Color Class

AWT Components

Swing Classes in JAVA

JDBC

Module 6: GUI Programming in JAVA

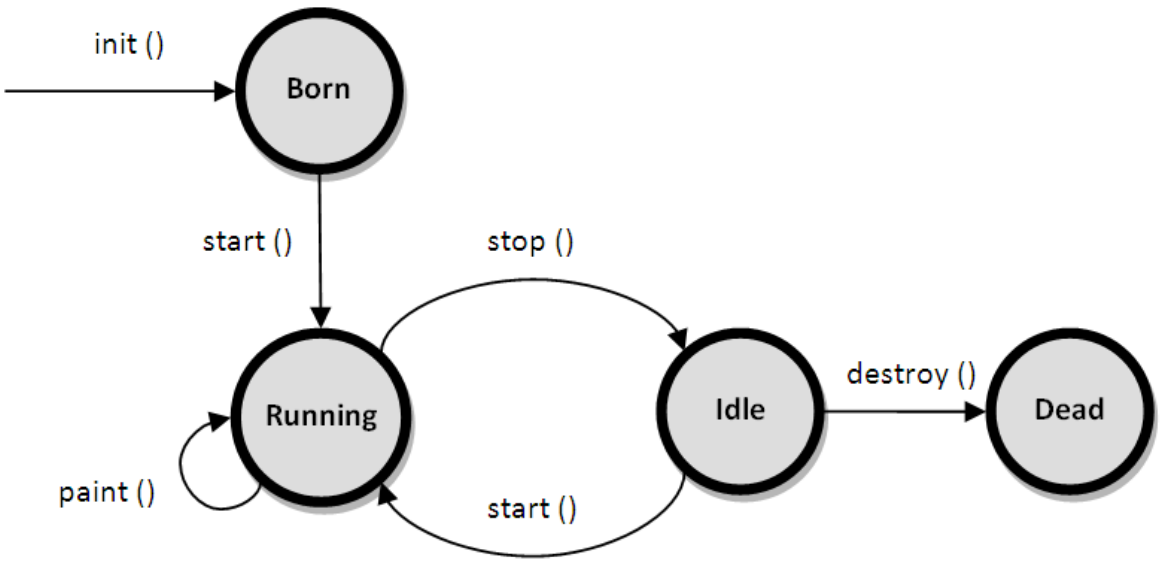
6.1 Applet and Applet Life Cycle:

Question 1: Explain applet life cycle with diagram
Answer:
Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet
There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many plateforms, including Linux, Windows, Mac Os etc.

The life cycle of an applet is as shown in the figure below:



As shown in the above diagram, the life cycle of an applet starts with *init()* method and ends with *destroy()* method. Other life cycle methods are *start(), stop()* and *paint()*. The methods to execute only once in the applet life cycle are *init()* and *destroy()*. Other methods execute multiple times.

init():
The init() method is the first method to execute when the applet is executed. Variable declaration and initialization operations are performed in this method.

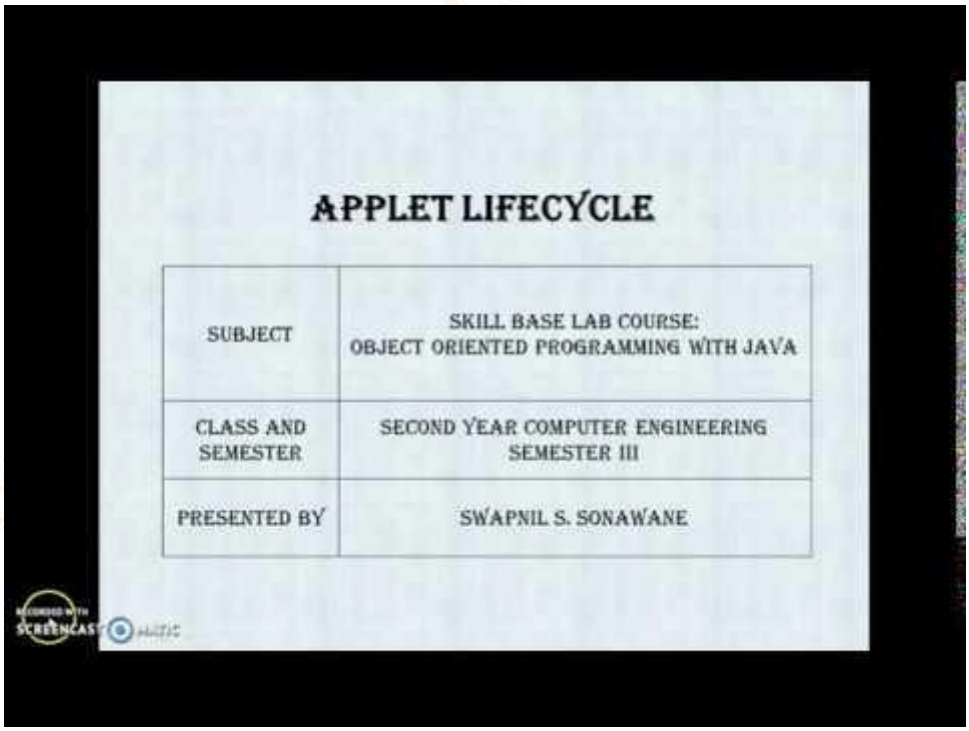
start():
The start() method contains the actual code of the applet that should run. The start() method executes immediately after the *init()* method. It also executes whenever the applet is restored, maximized or moving from one tab to another tab in the browser.

stop():
The stop() method stops the execution of the applet. The stop() method executes when the applet is minimized or when moving from one tab to another in the browser.

destroy():
The destroy() method executes when the applet window is closed or when the tab containing the webpage is closed. *stop()* method executes just before when destroy() method is invoked. The destroy() method removes the applet object from memory.

paint():
The paint() method is used to redraw the output on the applet display area. The paint() method executes after the execution of *start()* method and whenever the applet or browser is resized.

Video Link:



Web Links:
https://www.startertutorials.com/corejava/applet-life-cycle.html
https://www.geeksforgeeks.org/java-applet-basics/

6.2 Creating Applet:

Question 1: Write a program to print "Hello Applet" on applet viewer

Answer:

```
//applet1.java
import java.awt.*;
import java.applet.*;
public class applet1 extends Applet
{
String s;
public void init()
{
s="Hello Applet";
}
public void paint(Graphics g)
{
g.drawString(s,10,20);
}
}

//applet1.html
<html>
<body>
<applet code=applet1 width=200 height=300>
</applet>
</body>
</html>
```

Output:

6.3 Parameter Passing to Applet:

Question 1: Explain with example to pass parameters to an applet
Answer:
We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter()
Steps to accomplish this task -:

- To pass the parameters to the Applet we need to use the param attribute of <applet> tag.
- To retrieve a parameter's value, we need to use the getParameter() method of Applet class.

Example:

```
import java.applet.Applet;
import java.awt.Graphics;
public class UseParam extends Applet
{
public void paint(Graphics g)
{
String str=getParameter("msg");
g.drawString(str,50, 50);
}
}
```

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

6.4 Font and Color Class:

Question 1: Explain with example to pass parameters to an applet
Answer
Font Class:
The Font class states fonts, which are used to render text in a visible way.

Constructors:

Font();

Font(String name, int style, int size)

Name	Style
Serif	Font.BOLD
SansSerif	Font.ITALIC
Monospaced	Font.PLAIN
Arial	Font.BOLD|Font.ITALIC
TimesRoman	

To set a font we use Graphics class method:
setFont()

Color Class:

The Color class states colors in the default sRGB color space or colors in arbitrary color spaces identified by a ColorSpace.

Constructor:

Color(int r, int g, int b)

Creates an opaque sRGB color with the specified red, green, and blue values in the range (0 - 255).

To set a color we use Graphics class method:

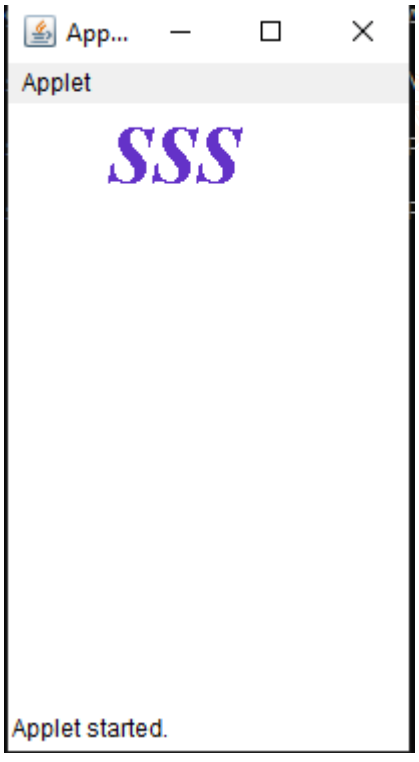
setColor()

Example:

```
import java.awt.*;
import java.applet.*;
public class applet3 extends Applet
{
public void paint(Graphics g)
{
Font f=new Font("TimesRoman",Font.BOLD|Font.ITALIC,40);
Color c=new Color(100,50,200);
g.setFont(f);
g.setColor(c);
g.drawString("SSS",50,40);
}
}
```

```
<html>
<body>
<applet code=applet3 width=200 height=300>
</applet>
</body>
</html>
```

Output:



Question 2: Explain Graphics class methods with example

Answer

Sr. No	Methods	Description
1.	drawString (String str, int x, int y)	It is used to draw the specified string.
2.	drawRect(int x, int y, int width, int height)	It draws a rectangle with the specified width and height.
3.	fillRect(int x, int y, int width, int height)	It is used to fill rectangle with the default color and specified width and height.
4.	drawOval(int x, int y, int width, int height)	It is used to draw oval with the specified width and height.
5.	fillOval(int x, int y, int width, int height)	It is used to fill oval with the default color and specified width and height.

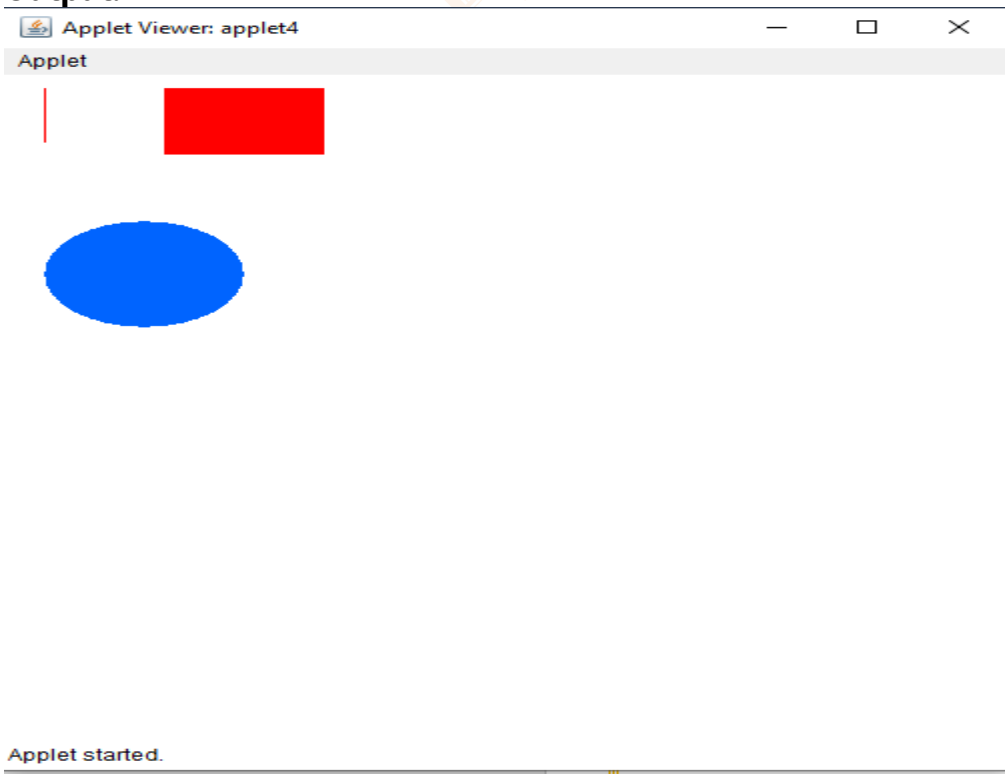
Sr. No	Methods	Description
6.	drawLine(int x1, int y1, int x2, int y2)	It is used to draw line between the points(x1, y1) and (x2, y2).
7.	drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)	It is used draw a circular or elliptical arc.
8.	fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)	It is used to fill a circular or elliptical arc.
9.	setColor(Color c)	It is used to set the graphics current color to the specified color.
10.	setFont(Font font)	It is used to set the graphics current font to the specified font.

Example:

```
import java.awt.*;
import java.applet.*;
public class applet4 extends Applet
{
public void paint(Graphics g)
{
Color c1=new Color(255,0,0);
g.setColor(c1);
g.drawLine(20,10,20,50);
g.fillRect(80,10,80,50);
Color c2=new Color(0,100,255);
g.setColor(c2);
g.fillOval(20,110,100,80);
}
}
```

```
<html>
<body>
<applet code=applet4 width=500 height=500>
</applet>
</body>
</html>
```

Output:



6.5 AWT Components:

Question 1: Explain different awt components with example

Answer

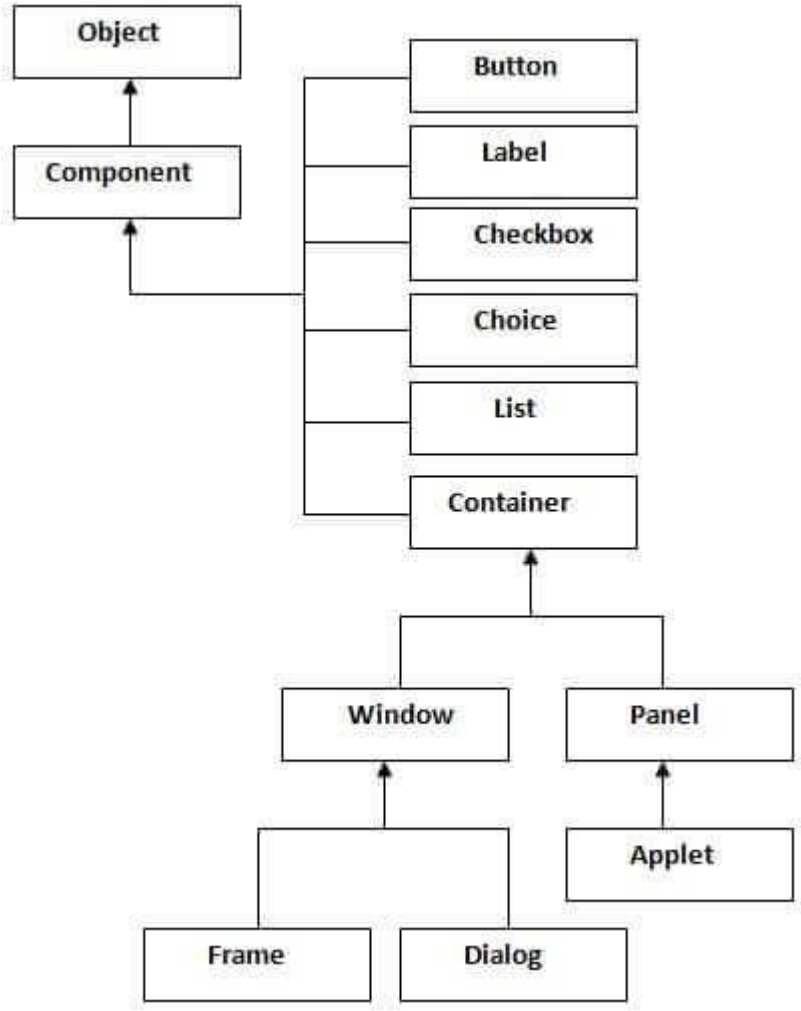
ava AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.

Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

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

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below.



Container
The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

Window
The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

Panel
The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

Frame
The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

Methods of Component Class:

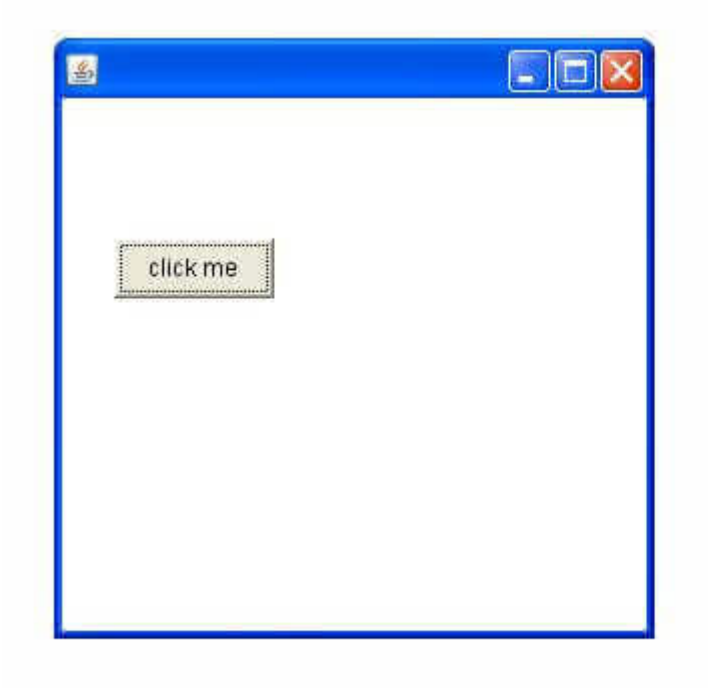
Method	Description
add(Component c)	Inserts a component on this component.
setSize(int width,int height)	Sets the size (width and height) of the component.
setLayout(LayoutManager m)	Defines the layout manager for the component.
setVisible(boolean status)	Changes the visibility of the component, by default false.

Program:

```
import java.awt.*;
class First extends Frame
{
First()
{
```

```
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
public static void main(String args[])
{
First f=new First();
}
}
```

Output:



6.6 SWING Classes in Java:

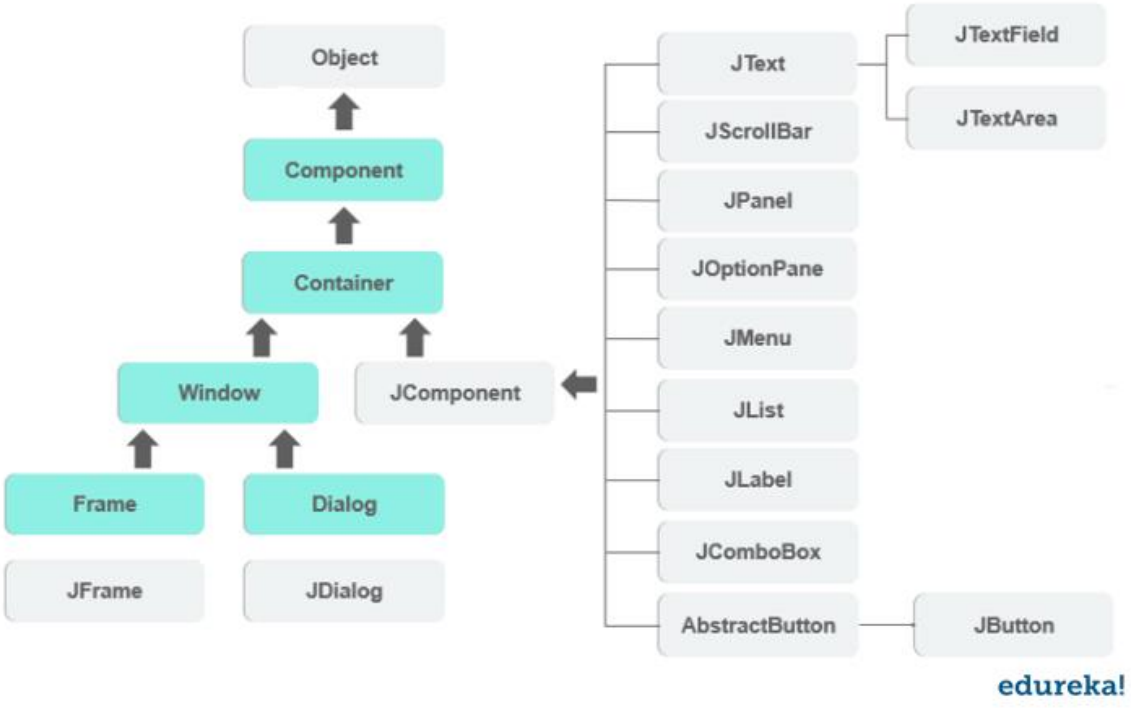
Question 1: Explain different swing classes in java

Answer

Swing in Java is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window based applications. It is a part of the JFC(Java Foundation Classes). It is build on top of the AWT API and entirely written in java. It is platform independent unlike AWT and has lightweight components.

It becomes easier to build applications since we already have GUI components like button, checkbox etc.

Java Swing Class Hierarchy



All the components in swing like JButton, JComboBox, JList, JLabel are inherited from the JComponent class which can be added to the container classes. Containers are the windows like frame and dialog boxes. Basic swing components are the building blocks of any gui application. Methods like setLayout override the default layout in each container. Containers like JFrame and JDialog can only add a component to itself. Following are a few components with examples to understand how we can use them.

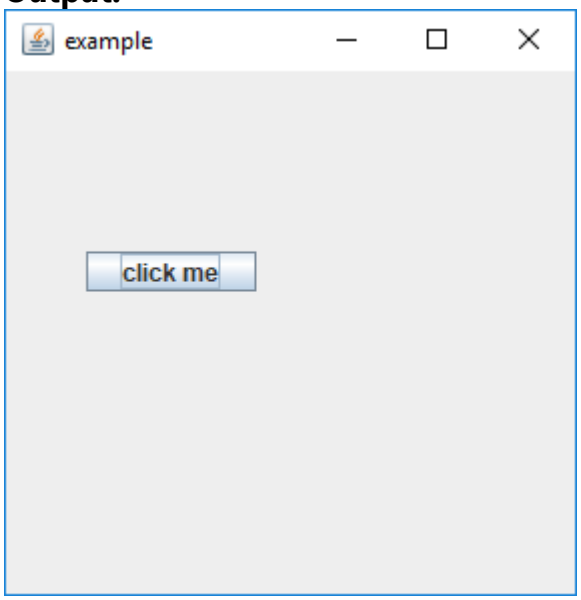
JButton Class

It is used to create a labelled button. Using the ActionListener it will result in some action when the button is pushed. It inherits the AbstractButton class and is platform independent.

Example:

```
1    import javax.swing.*;
2    public class example{
3    public static void main(String args[]) {
4    JFrame a = new JFrame("example");
5    JButton b = new JButton("click me");
6    b.setBounds(40,90,85,20);
7    a.add(b);
8    a.setSize(300,300);
9    a.setLayout(null);
10   a.setVisible(true);
11   }
12   }
```

Output:



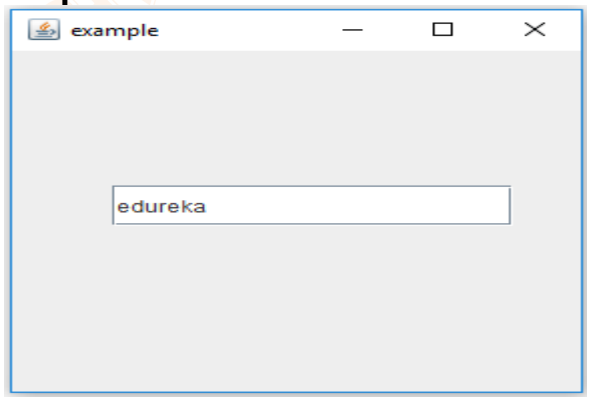
JTextField Class

It inherits the JTextComponent class and it is used to allow editing of single line text.

Example:

```
1    import javax.swing.*;
2    public class example{
3    public static void main(String args[]) {
4    JFrame a = new JFrame("example");
5    JTextField b = new JTextField("edureka");
6    b.setBounds(50,100,200,30);
7    a.add(b);
8    a.setSize(300,300);
9    a.setLayout(null);
10   a.setVisible(true);
11   }
12   }
```

Output:



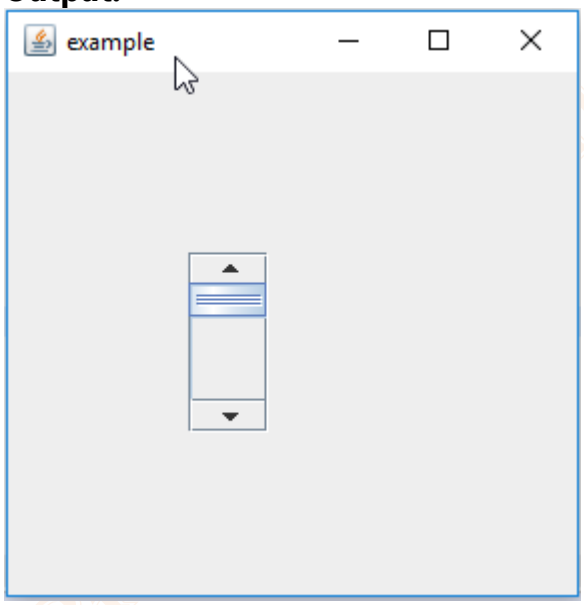
JScrollBar Class

It is used to add scroll bar, both horizontal and vertical.

Example:

```
1    import javax.swing.*;
2    class example{
3    example(){
4    JFrame a = new JFrame("example");
5    JScrollBar b = new JScrollBar();
6    b.setBounds(90,90,40,90);
7    a.add(b);
8    a.setSize(300,300);
9    a.setLayout(null);
10   a.setVisible(true);
11   }
12   public static void main(String args[]){
13   new example();
14   }
15   }
```

Output:

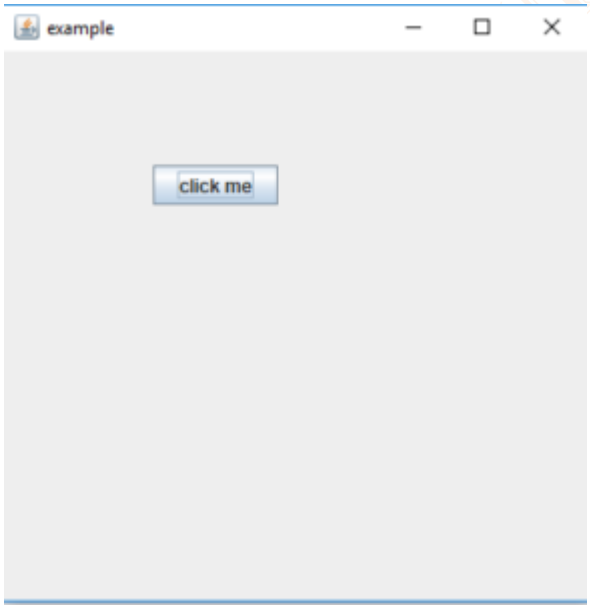


JPanel Class

It inherits the JComponent class and provides space for an application which can attach any other component.

```
1    import java.awt.*;
2    import javax.swing.*;
3    public class Example{
4    Example(){
5    JFrame a = new JFrame("example");
6    JPanel p = new JPanel();
7    p.setBounds(40,70,200,200);
8    JButton b = new JButton("click me");
9    b.setBounds(60,50,80,40);
10   p.add(b);
11   a.add(p);
12   a.setSize(400,400);
13   a.setLayout(null);
14   a.setVisible(true);
15   }
16   public static void main(String args[])
17   {
18   new Example();
19   }
20   }
```

Output:

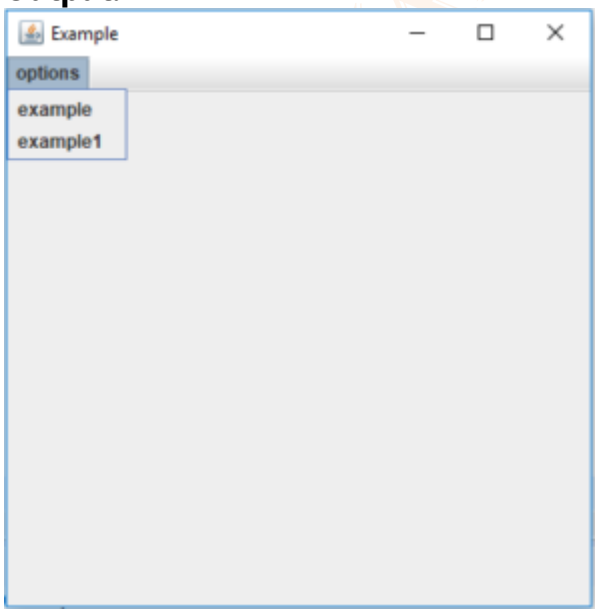


JMenu Class

It inherits the JMenuItem class, and is a pull down menu component which is displayed from the menu bar.

```
1    import javax.swing.*;
2    class Example{
3    JMenu menu;
4    JMenuItem a1,a2;
5    Example()
6    {
7    JFrame a = new JFrame("Example");
8    menu = new JMenu("options");
9    JMenuBar m1 = new JMenuBar();
10   a1 = new JMenuItem("example");
11   a2 = new JMenuItem("example1");
12   menu.add(a1);
13   menu.add(a2);
14   m1.add(menu);
15   a.setJMenuBar(m1);
16   a.setSize(400,400);
17   a.setLayout(null);
18   a.setVisible(true);
19   }
20   public static void main(String args[])
21   {
22   new Example();
23   }
24   }
```

Output:



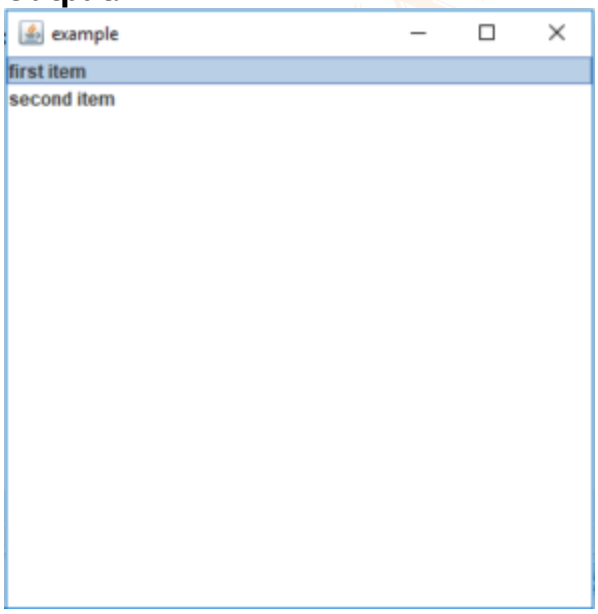
JList Class

It inherits JComponent class, the object of JList class represents a list of text items.

Programming & Frameworks Training

```
1    import javax.swing.*;
2    public class Example
3    {
4    Example(){
5    JFrame a  = new JFrame("example");
6    DefaultListModel<String> l = new DefaultListModel< >();
7    l.addElement("first item");
8    l.addElement("second item");
9    JList<String> b = new JList< >(l);
10   b.setBounds(100,100,75,75);
11   a.add(b);
12   a.setSize(400,400);
13   a.setVisible(true);
14   a.setLayout(null);
15   }
16   public static void main(String args[])
17   {
18   new Example();
19   }
20   }
```

Output:

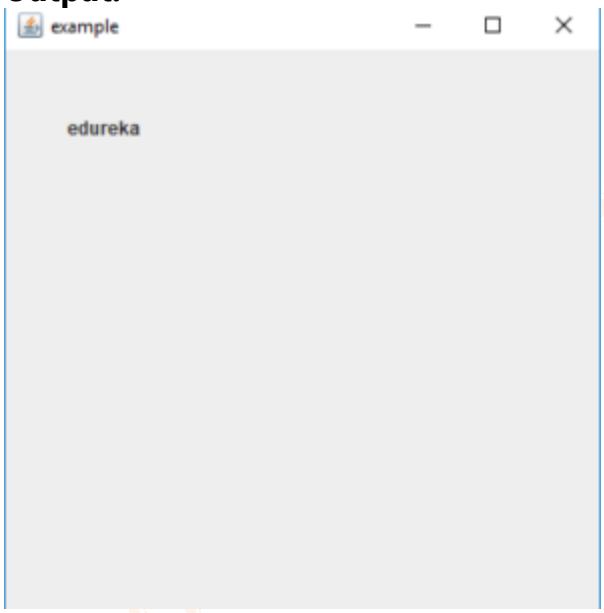


JLabel Class

It is used for placing text in a container. It also inherits JComponent class.

```
1    import javax.swing.*;
2    public class Example{
3    public static void main(String args[])
4    {
5    JFrame a = new JFrame("example");
6    JLabel b1;
7    b1 = new JLabel("edureka");
8    b1.setBounds(40,40,90,20);
9    a.add(b1);
10   a.setSize(400,400);
11   a.setLayout(null);
12   a.setVisible(true);
13   }
14   }
```

Output:

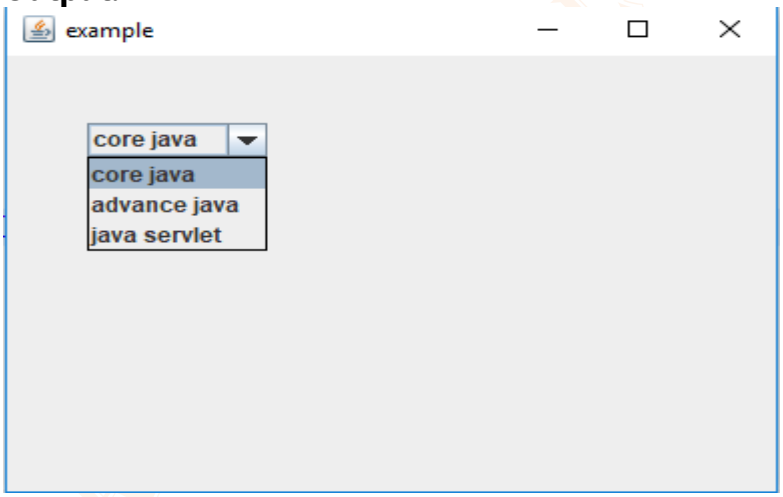


JComboBox Class

It inherits the JComponent class and is used to show pop up menu of choices.

```
1    import javax.swing.*;
2    public class Example{
3    JFrame a;
4    Example(){
5    a = new JFrame("example");
6    string courses[] = { "core java","advance java", "java servlet"};
7    JComboBox c = new JComboBox(courses);
8    c.setBounds(40,40,90,20);
9    a.add(c);
10   a.setSize(400,400);
11   a.setLayout(null);
12   a.setVisible(true);
13   }
14   public static void main(String args[])
15   {
16   new Example();
17   }
18   }
```

Output:



6.7 JDBC:

Question 1: Explain JDBC with example

Answer

JDBC stands for Java Database Connectivity, which is a standard Java API for database-independent connectivity between the Java programming language and a wide range of databases.

The JDBC library includes APIs for each of the tasks mentioned below that are commonly associated with database usage.

- Making a connection to a database.
- Creating SQL or MySQL statements.
- Executing SQL or MySQL queries in the database.
- Viewing & Modifying the resulting records.

We can use JDBC API to handle database using Java program and can perform the following activities:

- Connect to the database
- Execute queries and update statements to the database
- Retrieve the result received from the database.

JAVA Database Connectivity

It is used to access a database using a java program

Accessing database includes performing following operations on database:

1. Insert a new record in a database table
2. Delete a particular record from database table
3. Update a particular record from database table
4. Display the contents of database table

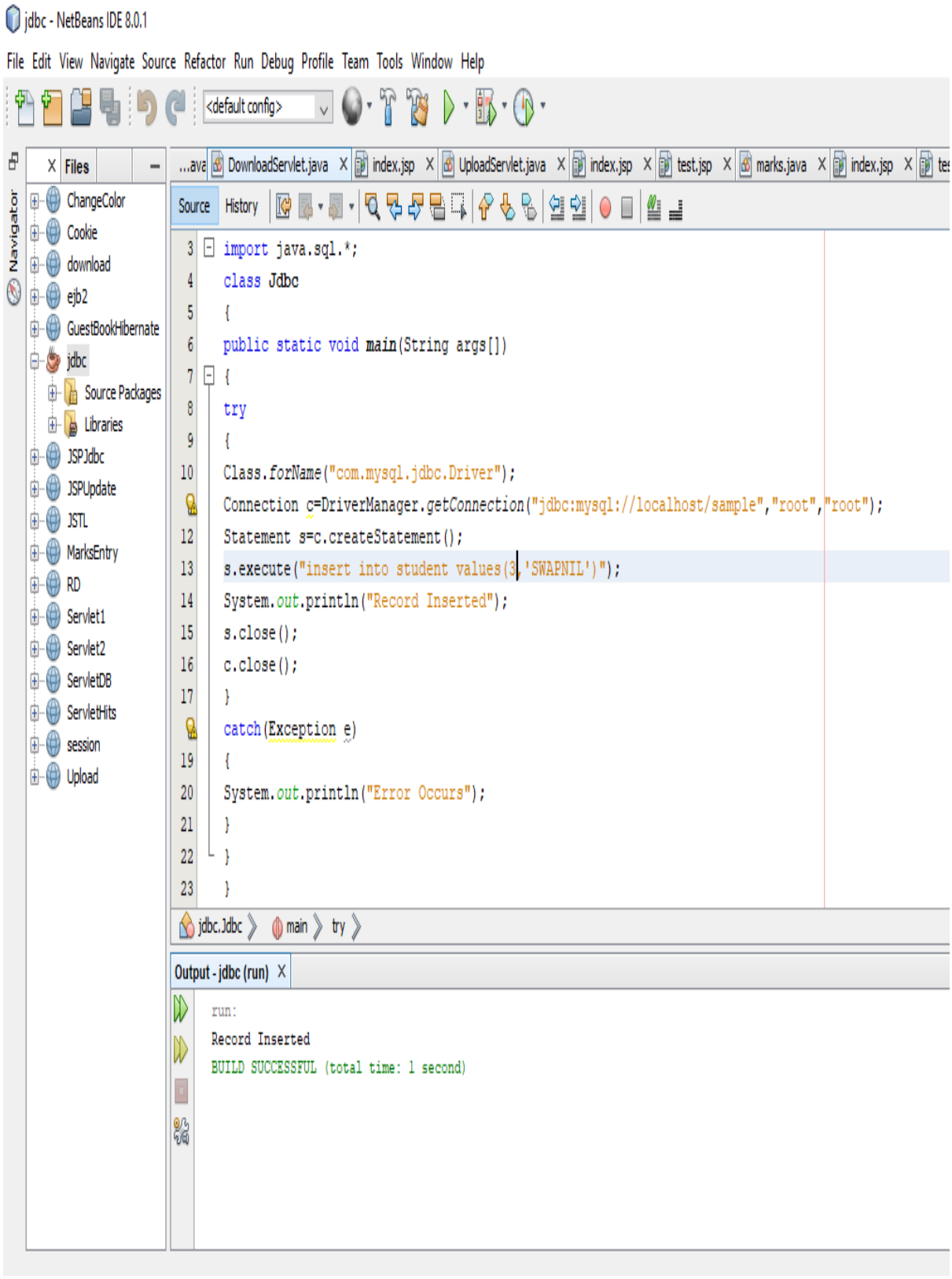
To perform these operations on database table we need to implement following steps in JDBC progrm:

Step 1	**Importing a package** import java.sql.*;
Step 2	**Loading a Driver** Class.forName("com.mysql.jdbc.Driver");
Step 3	**Establishing a Connection** Connection c=DriverManager.getConnection("jdbc:mysql://localhost/dbname", "username","password");
Step 4	**Creating a Statement** Statement s=c.createStatement();

Stem 5	**Executing Statement**
	s.execute();
	s.executeUpdate();
	ResultSet rs=s.executeQuery()

Step 6	**Retrieving a Result** ResultSet rs=s.getResultSet();
Step 7	**Closing a Connection** c.close();

Program:


jdbc - NetBeans IDE 8.0.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
<default config>
Files
ChangeColor
Cookie
download
ejb2
GuestBookHibernate
jdbc
Source Packages
Libraries
JSPJdbc
JSPUpdate
JSTL
MarksEntry
RD
Servlet1
Servlet2
ServletDB
ServletHits
session
Upload
DownloadServlet.java
index.jsp
UploadServlet.java
index.jsp
test.jsp
marks.java
index.jsp
Source
History
3 import java.sql.*;
4 class Jdbc
5 {
6 public static void main(String args[])
7 {
8 try
9 {
10 Class.forName("com.mysql.jdbc.Driver");
Connection c=DriverManager.getConnection("jdbc:mysql://localhost/sample","root","root");
12 Statement s=c.createStatement();
13 s.execute("insert into student values(3,'SWAPNIL')");
14 System.out.println("Record Inserted");
15 s.close();
16 c.close();
17 }
catch(Exception e)
19 {
20 System.out.println("Error Occurs");
21 }
22 }
23 }
jdbc.Jdbc
main
try
Output - jdbc (run)
run:
Record Inserted
BUILD SUCCESSFUL (total time: 1 second)

Output: